

ETICS

User Manual

ETICS SERVICE V. 2.0
(REVISION 2.0.1)

Copyright (c) Members of the ETICS Collaboration. 2007.

See <http://www.eu-etics.org/etics/partners/> for details on the copyright holders.

ETICS (“E-Infrastructure for Testing, Integration and Configuration of Software”) is a project funded by the European Union. For more information on the project, its partners and contributors please see <http://www.eu-etics.org>.

You are permitted to copy and distribute verbatim copies of this document containing this copyright notice, but modifying this document is not allowed. You are permitted to copy this document in whole or in part into other documents if you attach the following reference to the copied elements: "Copyright (C) 2007. Members of the ETICS Collaboration. <http://www.eu-etics.org>".

The information contained in this document represents the views of ETICS as of the date they are published. ETICS does not guarantee that any information contained herein is error-free, or up to date.

ETICS MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.

Content

1. INTRODUCTION	6
1.1. OVERVIEW.....	6
1.2. HOW ETICS REPRESENTS SOFTWARE INFORMATION	8
1.2.1. Modules: Projects, Subsystems and Components.....	8
1.2.2. Configurations.....	10
1.2.3. Platforms.....	12
1.2.4. Commands and Properties	13
1.2.5. Dependencies.....	15
1.2.6. Commands and Properties Search Order for Multiple Platforms	16
1.2.7. Property Inheritance	16
1.2.8. Locking.....	17
1.3. AUTHENTICATION, AUTHORIZATION AND ROLES	17
2. ETICS PORTAL.....	20
2.1. OVERVIEW.....	20
2.2. LAYOUT.....	20
2.3. PANEL – MYETICS.....	20
2.3.1. Welcome.....	21
2.3.2. My submissions.....	21
2.4. PANEL – BUILD AND TEST SYSTEM.....	21
2.5. PANEL – REPOSITORY	21
2.6. PANEL – ADMINISTRATION.....	21
2.7. PANEL – EXTERNALS	22
2.8. PANEL – PROCESS NEW EXTERNAL COMPONENT REQUEST.....	22
3. THE ETICS BUILD AND TEST WEB APPLICATION.....	24
3.1. OVERVIEW.....	24
3.2. LAYOUT.....	25
3.3. ENABLING SECURITY	26
3.4. CERTIFICATE REGISTRATION	27
4. THE ETICS COMMAND-LINE CLIENT.....	29
4.1. OVERVIEW.....	29
4.2. HOW TO INSTALL THE ETICS CLIENT	29
4.3. HOW TO CONFIGURE THE ETICS CLIENT.....	30
4.4. WORKSPACES	32
4.5. ETICS COMMANDS AND LIBRARIES.....	32
4.6. ENABLING SECURITY	33
5. BROWSING MODULES AND CONFIGURATION INFORMATION	34
5.1. OVERVIEW.....	34
5.2. HOW TO BROWSE WITH THE WEB APPLICATION.....	34
5.2.1. Selecting a Project.....	34
5.2.2. Browsing a Project, Subsystems and Components	34
5.2.3. Searching Configurations.....	36
5.2.4. Adding configurations to the Workspace.....	37
5.2.5. Browsing Configurations and Sub-Configurations	37
5.2.6. Listing supported Platforms	38
5.2.7. Viewing Commands, Properties and Dependencies	39
5.3. HOW TO BROWSE WITH THE COMMAND-LINE CLIENT.....	40
5.3.1. How to Get a Project.....	40
5.3.2. How to Checkout and Browse Configurations.....	40

5.3.3.	How to Show the Structure of Modules and Configurations?	41
5.3.4.	Modules, Platforms, Users, Roles and Other Objects	42
6.	CHECKING OUT PROJECT, SUBSYSTEMS AND COMPONENTS.....	43
6.1.	OVERVIEW	43
6.2.	SOURCE CODE OR BINARY PACKAGES?	43
6.3.	THE ETICS-CHECKOUT COMMAND	43
6.4.	HOW TO CHECKOUT SOURCE CODE OR BINARY PACKAGES	45
6.4.1.	Merging Configurations	46
6.4.2.	Updating Configurations	46
6.4.3.	Forcing Checkout	47
6.4.4.	Permanent and Volatile Repositories	47
7.	EDITING PROJECTS	49
7.1.	OVERVIEW	49
7.2.	HOW TO EDIT WITH THE WEB APPLICATION	49
7.2.1.	Editing Modules	49
7.2.2.	Editing Configurations	51
7.2.3.	Editing Sub-Configuration Relationships	55
7.2.4.	Attaching Platforms to Configurations	57
7.2.5.	Editing Commands	58
7.2.6.	Editing Properties and Environment	58
7.2.7.	Editing Dependencies	59
7.3.	HOW TO EDIT WITH THE COMMAND-LINE CLIENT	60
7.4.	THE ETICS-MODULE COMMAND	63
7.4.1.	How to prepare a module	64
7.4.2.	How to add a module	65
7.4.3.	How to modify a module	66
7.4.4.	How to rename a module	67
7.4.5.	How to remove a module	67
7.5.	THE ETICS-CONFIGURATION COMMAND	68
7.5.1.	How to prepare a configuration	69
7.5.2.	How to add a configuration	72
7.5.3.	How to clone a configuration	73
7.5.4.	How to modify a configuration	73
7.5.5.	How to remove a configuration	74
7.5.6.	How to rename a configuration	74
7.5.7.	How to lock a configuration	75
7.6.	THE ETICS-COMMIT COMMAND	75
8.	TAGGING CONFIGURATIONS	76
8.1.	OVERVIEW	76
8.2.	THE ETICS-TAG COMMAND	76
8.2.1.	How to Tag a Configuration	77
8.2.2.	How to Tag a Configuration Tree	77
9.	BUILDING CONFIGURATIONS	79
9.1.	OVERVIEW	79
9.2.	THE ETICS-BUILD COMMAND	79
9.2.1.	How to Build a Configuration	80
9.2.2.	Build Targets	80
9.2.3.	Forcing Build Execution	81
9.3.	THE ETICS PACKAGING SYSTEM	81
9.3.1.	How to Pass Installation Instructions to the Packager	86
9.4.	THE ETICS PUBLISHER	87



9.5.	BUILD REPORTS	89
10.	TESTING CONFIGURATIONS	90
10.1.	OVERVIEW	90
10.2.	THE ETICS-TEST COMMAND	90
10.2.1.	<i>How to Test a Configuration</i>	<i>91</i>
10.2.2.	<i>Test Targets.....</i>	<i>91</i>
10.3.	TEST REPORTS.....	92
11.	SUBMITTING REMOTE BUILDS AND TESTS TO THE ETICS SYSTEM	93
11.1.	SUBMITTING BUILDS AND TESTS USING THE WEB APPLICATION	93
11.2.	SUBMITTING BUILDS AND TESTS USING THE COMMAND-LINE CLIENT	93
11.2.1.	<i>How to submit a remote build job</i>	<i>99</i>
11.2.2.	<i>How to submit a remote test job.....</i>	<i>99</i>
12.	ACCOUNT INFORMATION	100
13.	REPOSITORY	102
13.1.	VOLATILE STORAGE AREAS.....	102
13.2.	REGISTERED STORAGE AREA.....	103
13.3.	REPOSITORY WEB APPLICATION.....	103
14.	ANALYSING BUILD AND TEST REPORTS	108
14.1.	STANDARD REPORTS	108
14.2.	THE ETICS-GET-REPORT COMMAND	113
14.3.	THE ETICS-GET-RUNDIR COMMAND	114
14.4.	SPECIFIC REPORTS	114
15.	USER AND MODULE ADMINISTRATION	120
15.1.	THE ETICS ADMINISTRATION APPLICATION	120
15.1.1.	<i>Layout.....</i>	<i>120</i>
15.1.2.	<i>Manage projects</i>	<i>121</i>
15.1.3.	<i>Manage users.....</i>	<i>122</i>
15.1.4.	<i>Manage user permissions.....</i>	<i>123</i>
15.2.	ADMINISTRATION WITH THE COMMAND-LINE CLIENT.....	126
15.2.1.	<i>Manage users.....</i>	<i>126</i>
15.2.2.	<i>Manage user permissions.....</i>	<i>127</i>
16.	CLIENT PLUGIN FRAMEWORK	129
16.1.	OVERVIEW	129
16.2.	PLUGIN SPECIFICATION	130
A.	PLUGIN SAMPLES	132
B.	PROPERTIES	141

1. INTRODUCTION

This user manual describes the ETICS Service v2.0.

1.1. Overview

ETICS stands for eInfrastructure for Testing, Integration and Configuration of Software. ETICS is an on-line collaborative service for managing small and large software projects by managing their configuration, enforcing quality standards, building packages and testing them in environments as close as possible to real-world infrastructures.

The ETICS tools can be used to run builds and tests on users' computers, but also and especially to submit remote builds and tests on multiple platforms at the same time.

The process of building and testing software with ETICS goes through three phases:

1. **Configuration:** this is the process of registering the software components to be built and tested with the ETICS system; creating configurations and attaching version control commands, build commands, test commands, properties, environment variables and dependencies. Functionalities required to accomplish this task are provided by the Build and Test Web Application and Command-Line Client mediating the access to the underlying Build and Test Service
2. **Local build/test:** once configuration information is registered in the system, the ETICS command-line client allows performing local builds and tests on a user machine. This step allows the user to validate the configuration information and make sure the components build and pass tests as expected and produce the expected results. In the case of building, the outcome of this procedure is a set of packages in various formats (typically tarballs and RPMS on Linux systems) and a set of log files, test and metrics reports. When performing system tests, a set of log files, test and metrics reports is also generated.
3. **Remote build/test:** once the components have been verified to build and/or test successfully locally, it is possible to submit and schedule builds and tests on a pool of worker nodes (i.e. remote machines) providing support for different platforms. This step allows running the builds and tests on a variety of platforms to validate portability of the code and generate a number of static and dynamic test reports under various conditions. The remote execution is implemented using the Metronome engine. Once on a worker node the build and test procedures are performed by the ETICS Command-Line Client, follow the exact same sequence as a local build and test procedure.

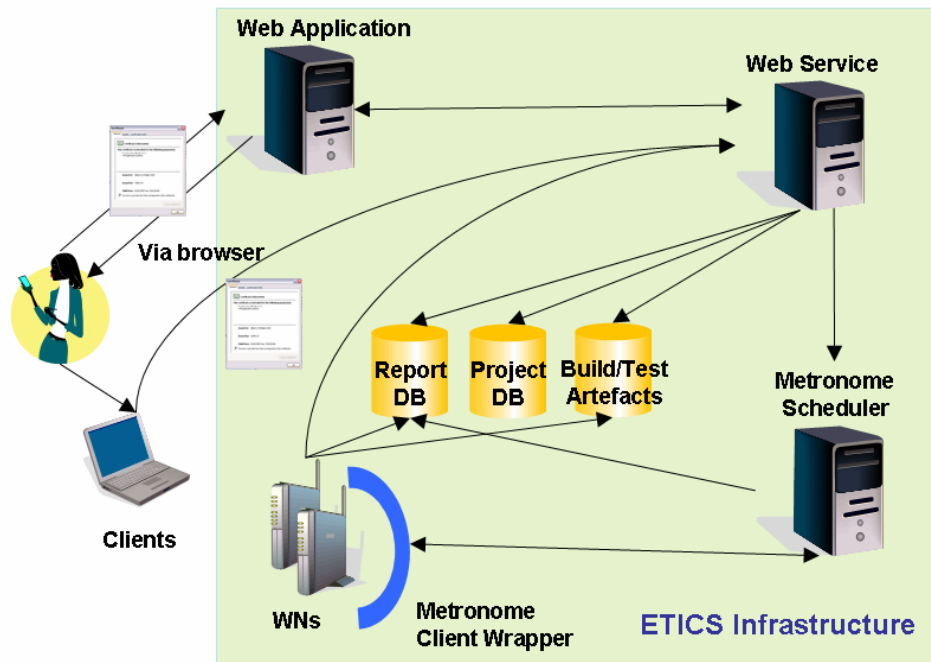


Figure 1: The ETICS Architecture

The ETICS Services includes of the following main components (Figure 1):

- **The ETICS Client:** a set of command line tools to access the ETICS services. The tools are written in Python and are currently available on most platforms¹ with Python >= 2.2
- **The ETICS Web Application:** a web application to manage the configuration of projects registered in the ETICS metadata store
- **The ETICS Web Service:** the web service is the business core of the system. It handles requests from the command-line client and the web application, processes requests and forwards build/test jobs requests to the Metronome engine
- **The Metronome Execution Engine:** the Metronome engine uses the Condor system to execute jobs on dedicated worker nodes. The worker nodes operates on different platforms to provide a multiplatform build/test environment
- **The ETICS Database:** all configuration information is stored in the ETICS database, which allows creating reproducible builds, setting dependencies between projects and packages and creating quality controlled software packages according to stored policies. In this manual, we refer to the data stored in the database as the *metadata store*.
- **The ETICS Repository:** software packages and test data generated by running build and test jobs in ETICS system can be automatically uploaded to the software repository, where they can be accessed by users or by the ETICS system itself. Remote build and test reports can also be browsed from the ETICS Repository or locally with a standard web browser.

¹ At the time of releasing this document, the client doesn't work on Windows.

1.2. How ETICS Represents Software Information

Software projects need to be registered in the ETICS metadata store in order to use the build and test service. Before using ETICS it is recommended to get familiar with a few basic concepts used by the system. The objects described in the following paragraphs are part of the ETICS data model.

1.2.1. Modules: Projects, Subsystems and Components

Software projects are known in ETICS as *Projects*. A Project can be considered as a high level container for software components and it's a good place to define properties and policies that must be applied by default to all components, as will be explained later.

A project can be further split in *Subsystems* and *Components*. A subsystem represents a logical partition of the software architecture providing some subset of the project functionality. As for the project, subsystems can normally be considered containers for software components with special properties and policies that must be applied by default to all components of the subsystem. However, the subsystem can be a software component in its own right and normally can be used to produce meta-packages.

Components represent even smaller unit of functionality in the project architecture. A component can belong to the project or to an individual subsystem in the project. Normally they are used to generate individual software packages or small families of packages. Figure 2 shows the relationships among project, subsystem and component objects.

A subsystem must belong to one and only one project and a component must belong to one and only one project or subsystem. Project, subsystem and component objects are generically referred to as *Module*.

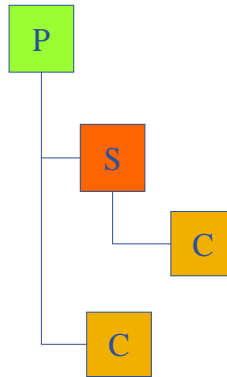


Figure 2: The Project, Subsystem and Component objects

Project

The project parameters are:

Table 1: Project parameters

Parameter	Description	Mandatory
name	The name of the project set. It must be unique in the ETICS metadata store	Yes
displayName	A friendlier name for the project set. It doesn't need to be unique	No
description	The project set description	No
homepage	The home page of the project	No
logo	The URL of the project logo in a format suitable to be used on a	No

repository	web page The default URL of the repository storing packages from this project (can be overridden by subobjects)	No
vcsroot	The default root of the version control system storing source code for this project (can be overridden by subobjects)	No
vendor	The project owner (organization, company, individual)	No
modifyDate	The date of last modification	No
createDate	The date of creation	No

Subsystem

The subsystem parameters are:

Table 2: Subsystem parameters

Parameter	Description	Mandatory
name	The name of the subsystem set. It must be unique in the ETICS metadata store	Yes
displayName	A friendlier name for the subsystem set. It doesn't need to be unique	No
description	The subsystem set description	No
repository	The default URL of the repository storing packages from this subsystem (can be overridden by subobjects)	No
vcsroot	The default root of the version control system storing source code for this subsystem (can be overridden by subobjects)	No
vendor	The subsystem owner (organization, company, individual)	No
modifyDate	The date of last modification	No
createDate	The date of creation	No

Component

The component parameters are:

Table 3: Component parameters

Parameter	Description	Mandatory
name	The name of the component set. It must be unique in the ETICS metadata store	Yes
displayName	A friendlier name for the component set. It doesn't need to be unique	No
description	The component set description	No
packageName	The name of the package produced by this component if different from the component name	No
homepage	The home page of the project	No
download	The URL of the download page for this component if taken directly in binary format from the vendor	No
licenceType	The license to which this component is subject	No
repository	The URL of the repository storing packages from this component	No
vcsroot	The root of the version control system storing source code for this component	No
vendor	The subsystem owner (organization, company, individual)	No
modifyDate	The date of last modification	No
createDate	The date of creation	No

1.2.2. Configurations

A *Configuration* in ETICS is the set of information representing a specific version of a module. For example, one can compare the ETICS modules to CVS modules and the configurations to CVS tags and branches.

A configuration must be attached to one and only one module (project, subsystem or components), but each module can have one or many configurations, like in CVS a module always has the HEAD branch, but can have many other tags and branches. Configurations are organized in a hierarchy that mirrors that of the corresponding modules. Therefore if a project contains some subsystems and each subsystem contains some components, a project configuration can contain subsystem configurations and the subsystem configurations can each contain component configurations (Figure 3).

However, noted that a configuration tree doesn't have to contain configurations for all objects in the corresponding module tree. The module tree in a project represents the project structure, but a configuration tree is a particular version of the project and can contain only a subset of the configurations. Projects and subsystems configuration trees can contain any combination of the available configurations, provided they do not contain more than one configuration of the same module (this requirement is relaxed for external dependencies, since it is possible to have different versions of the same dependency used by different components in the same project).

In the ETICS system the configurations are the objects on which the build and test operations are performed. When a project configuration is built, all subsystem configurations in its tree are automatically built unless differently specified. In the same way, when a subsystem configuration is built, all component configurations in its tree are built.

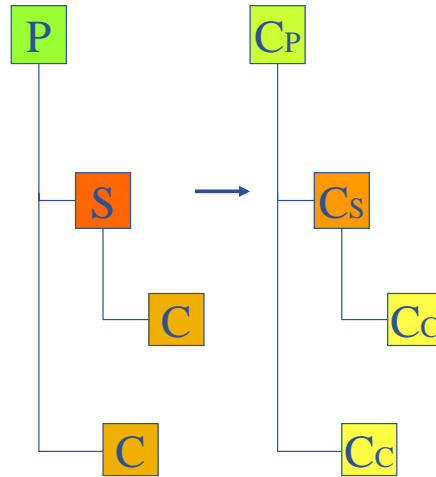


Figure 3: The ETICS Modules and Configurations

The configuration parameters are:

Table 4: Configuration parameters

Parameter	Description	Mandatory
name	The name of the configuration set. It must be unique in the ETICS metadata store	Yes
displayName	A friendlier name for the configuration set. It doesn't need to be unique	No
description	The configuration description	No
age	The configuration age number	No
majorVersion ¹	The configuration major version number	Yes
minorVersion	The configuration minor version number	No
revisionVersion	The configuration revision version number	No
path	The path where the package file can be downloaded from relative to the repository	No
profile	A comma-separated list of profiles	No
status	The configuration status (alpha, beta, RC, etc)	No
tag	The tag string of this configuration in the Version Control System hosting the module	No
modifyDate	The date of last modification	Auto
createDate	The date of creation	Auto

In addition a configuration inherits the following parameters from its corresponding module (i.e. project, subsystem or component):

¹ Standard version information can be represented in the form of: <majorVersion>.<minorVersion>.<releaseVersion>-<age>

Table 5: Inherited module parameters

Parameter	Description
moduleName	The name of the parent module
moduleDescription	The description of the parent module
licenseType	The license of the parent module
projectName	The name of the project to which the parent module belongs

1.2.3. Platforms

The information required to build or test a configuration may depend on the platform where the operation is executed. In order to allow the correct execution on multiple platforms, the ETICS metadata store defines platform objects representing particular combinations of OS types, architectures and compilers.

Each platform is identified as a string of the form:

```
<os>_<arch>_<compiler>
```

For example the following platforms are currently defined:

```
slc3_ia32_gcc323
slc4_ia32_gcc346
slc4_x86_64_gcc346
win32_vc71
```

The platform is automatically detected when running client commands, but it can be overridden for special reasons as described in Chapter 9, for example if your working platform doesn't exactly match one of the predefined platforms.

All commands and properties (described in the following sections) are attached to configurations via one or more platforms. Each configuration can have separate sets of commands and properties for each platform. In addition, a special *default* platform is defined in the system to which default sets of commands and properties can be attached to be used whenever a platform-specific value is not defined (e.g. when a specific behaviour for a given platform is not required).

The platform parameters are:

Table 6: Platform parameters

Parameter	Description	Mandatory
Name	The name of the platform set. It must be unique in the ETICS metadata store	Yes
displayName	A friendlier name for the platform set. It doesn't need to be unique	No
description	The platform set description	No
architecture	The CPU architecture (ia32, x86_64, etc)	No
family	The OS family (slc4, win32, etc)	No
vendor	The OS vendor name (Red Hat, Microsoft, etc)	No
age	The OS age	No

majorVersion	The OS major version number	No
minorVersion	The OS minor version number	No
revisionVersion	The OS revision version number	No

1.2.4. Commands and Properties

In order to build and test a configuration, the proper commands and properties must be defined. Commands and properties are attached to a configuration by platforms. There are three different sets of commands and two types of properties (i.e. ETICS properties and environment variables):

VCS Commands

The VCS Commands are used to manage code stored in a Version Control System (VCS), for example CVS, Subversion or ClearCase. The commands are defined as:

Table 7: VCS Command parameters

Command name	Description	Mandatory
description	The command set description	No
tag	Command to tag code	No
commit	Command to commit code ¹	No
checkout	Command to commit code	No
branch	Command to branch ²	No

More information about the VCS Commands can be found in Chapter 6 (“Checking out Project, Subsystems and Components”).

Build Commands

The Build Commands are used to build the code, generate documentation and packages and publish the build artefacts in a standard location and format. The commands are defined as:

Table 8: Build Command targets

Command name	Description	Mandatory
description	The command set description	No
clean	Command to clean	No
init	Command to perform initialization operations that depend on the build system structure (for example fetching files from other modules). This command cannot be used in packaging scripts (like the RPMS spec file)	No
configure	Command to perform initialisation operations that do not depend on the build system structure (e.g. running the ./configure script as part of the RPM post-install step, which the ETICS packager automatically inserts in the RPM generated spec file)	No
checkstyle	Command to perform code checking operations (coding	No

¹ Not supported in the current release

² Not supported in the current release

	conventions)	
Compile	Command to compile code	No
Test	Command to perform static tests like unit tests or coverage tests	
Doc	Command to generate documentation	No
packaging	Command to generate distribution packages	No
prepublish	Commands to perform steps to be executed before publishing the build artefacts (collecting additional files or applying transformations, etc)	No
Publish	Command to publish build artefacts to standard distribution formats	No
postpublish	Commands to perform steps to be executed after publishing the build artefacts (cleaning or publishing to custom locations)	No
Install	Command to install the package	No (however, the ETICS Packager ¹ requires this target to be defined)

More information about the Build Commands can be found in Chapter 9 (“Building Configurations”).

Test Commands

The Test Commands are used to perform dynamic (integration, system) tests on software packages and generate test report. The commands are defined as:

Table 9: Test Command parameters

Command name	Description	Mandatory
description	The command set description	No
clean	Command to clean	
init	Command to perform initialization operations before compiling	No
test	Command to execute the tests	No

More information about the Test Commands can be found in Chapter 10 (“Testing Configurations”).

Properties

Properties are key/value pairs that can be attached to a configuration for a specific platform (or the default platform). The properties can then be used in any command or other properties as `${property-name}`. The ETICS system defines a number of built-in properties (see Appendix B: Properties for details on the built-in properties) that are always available when executing the commands, but users can define any number of custom properties or in most cases override the values

¹ For details on the ETICS Packager, see section 9.3 - The ETICS Packaging System

of the built-in properties. The properties are inherited and propagated following the algorithm described in section 1.2.7.

Environment variables

Environment variables are key/value pairs that can be attached to a configuration for a specific platform (or the default platform). The environment variables are appended to the execution environment and are visible to the ETICS commands and any script or program invoked by the commands.

Normally, environment variables are simply (re)set the specified by the value in the configuration. However, if the environment variable ends with the string 'PATH', then if the environment variable is already defined, the new value is pre-pended to the existing value. This allows the user to *accumulate* the values of special environment variables, for example PYTHONPATH or PATH.

1.2.5. Dependencies

It is possible to define dependencies between configurations, not only within the same subsystem or project, but also across subsystems and projects. Also the dependencies are attached to configurations by platform. Therefore it is possible to have different dependencies on different platforms for the same configuration.

There are two types of dependencies: static and dynamic.

Static dependencies

Static dependencies are fixed, named dependencies between two configurations. They are specified by name and are not affected by build properties or policies.

Dynamic dependencies

Dynamic dependencies are defined between a configuration and a module. The specific configuration (version) of the module to be used as dependency is dynamically defined at checkout using properties and may change if the same parent configuration is used in different configuration trees. This allows for example to set a dependency on a module, but always get whatever default configuration is defined at the project level without having to change the dependency explicitly when dependencies are upgraded.

The dynamic dependency resolution is performed according to the following rules:

1. By setting a property of the format:

`<module-name>.DEFAULT`

at any level or above the component having the dependency (for example this can be done at the project level so that the value is used by all configurations in the project). This method works for all dependencies both internal and external to the current project and it's the normal way of setting dynamic dependencies on external components.

2. By using the current configuration tree. For example assume that `SubsystemConfigA` contains two components, `ComponentConfigB` and `ComponentConfigC`, and that `ComponentConfigB` depends dynamically on `ComponentC`. The value of the dependency will be set automatically by the system to `ComponentConfigC`, since it belongs to the current configuration tree. This method works only for dependencies internal to the project

and it's the normal way of setting dynamic dependencies among components within the same project and configuration tree.

As of ETICS v2.0, the default values for dynamic dependencies have been removed. Instead, users can either lock their configuration (see details in section [XX](#)), or specify a *constraint by version*. Constraints by version allow the user to define a constraint for the dynamic resolution of a dependency, in terms of version information and optionally an inequality. The system then uses the existing above mentioned dynamic dependency mechanisms to resolve the dynamic dependency, and if a constraint by version is defined, it then tests if the resolution is compliant the user defined constraint.

In addition, both static and dynamic dependencies have a scope and can be **build-time**, **run-time** or both. Build-time dependencies are used during build operations and are automatically built if necessary, but are not added as dependencies for distribution packages (for example in RPMS spec files). Run-time dependencies are not used during a build, but are used to define dependencies in the distribution packages (again, for example in RPMS spec files).

1.2.6. Commands and Properties Search Order for Multiple Platforms

When commands and properties are defined for different platforms, the system looks for them in a specific order, which is slightly different for commands and properties

Commands

When a configuration is built or tested, the system first looks for commands sets defined for the current working platform (for example `slc3_ia32_gcc323`), if a command set is not defined it looks for an equivalent command set defined for the *default* platform, if none is defined, the command set is not executed. The command sets are taken as complete sets. For example if a build command set is defined for “`slc3_ia32_gcc323`”, and it defines the `init` target, but not the `configure` target, then `configure` is not executed even if the command set attached to the default platform defines it.

Properties

When a configuration is built or tested, the system first looks for properties defined for the default platform, then for properties defined for the current working platform. The resulting properties set include both sets of properties. If a property is defined for both the current working platform and the default platform, the platform-specific one is used.

1.2.7. Property Inheritance

Property processing in ETICS includes inheritance from the parent properties, as well as propagation of properties from children configurations and dependencies, providing both a top/bottom and bottom/up capabilities.

Each configuration in ETICS inherits the properties of its parent (e.g. project and possibly subsystem for a component configuration), which means that any property defined *above* a configuration is available to the configuration, where the property definition closest to the configuration overrides properties further defines. For example, say a component configuration is set as a child configuration of a subsystem and project configurations (i.e. parent configurations). If both parent configurations define the same property, when processing the component configuration, the value will be taken from the subsystem configuration, since it is *closest* to the component configuration than the project configuration.

This mechanism allows users to set properties at project level (e.g. debug level, default installation location) for all configurations to use, while allowing subsystems and component configurations to overwrite these default project settings.

The bottom/up propagation of property works as follows. If a configuration has children configurations and dependencies, all of their properties will be available to it. To avoid name clashing (i.e. in the case that different configurations define the same properties), all properties defined for the children configuration and dependencies are namespace qualified with the module name of their configuration. For example, say a component configuration `MyComponent` has a dependency on another component configuration with the module name called `MyDependency`, and in turn this dependency defines a property called `myProperty`, then this property will be available to `MyComponent` as `MyDependency.myProperty`.

Finally, the user can mix the two concepts, where a property can be defined at a higher level so that they are accessible only by a particular component down the hierarchy by qualifying the property name with the module name of the particular component. For example, assume that the configuration of project `MyProject` contains the configuration `MySubsystem`, which contains the configuration `MyComponent`. If the property `myProperty` is defined for the project it will be accessible by the project, the subsystem and the component. If the property `MyComponent.myProperty` is defined at the project level, it will only be accessible by the configuration `MyComponent`. However, note that `MyComponent.myProperty` will be available to all children in the project hierarchy.

The environment variables follow a similar algorithm, with the exception that no namespace qualification takes place. As mentioned in section 1.2.4, normally, environment variables are simply (re)set the specified by the value in the configuration. However, if the environment variable ends with the string 'PATH', then if the environment variable is already defined, the new value is pre-pended to the existing value. This allows the user to *accumulate* the values of special environment variables, for example `PYTHONPATH` or `PATH`.

1.2.8. Locking

This feature introduced in ETICS v2.0 allows users to make a configuration self consistent. By self consistent we mean that a given "locked" configuration will contain all the required information (i.e. properties, environment variables, dependencies resolution) in order to be built/tested, independently of the context in which it is built or tested. This will guarantee the reproducibility of those configurations across over time.

Several constraints come with locking configurations. In order to guarantee the above mentioned goal, locked configurations graphs can only contain locked configurations. In other words, in order to lock a project or a subsystem configuration with sub-configuration, or if any configuration has dependencies, the sub-configurations and dependencies must also be locked. This means that in order for a configuration graph to be locked in a single operation, the user must have appropriate privileges to lock all the configurations part of the graph (only Administrators, Release Managers and Developers can lock configurations). Failure to fulfil this requirement, the locking procedure will fail. Once a configuration has been locked, no further modifications are allowed to this configuration. This implies that the users will have to create a new copy (clone) of the locked configuration if he/she wants to make modifications.

Finally, only the artefacts generated from locked configurations can be registered in the permanent storage of the repository. This means that users must lock their configurations prior to registering artefacts in the registered part of the repository. Since artefacts can only be registered once, it is important that the configurations used to build a given artefact do not change once it has been registered, this is guaranteed by the locking mechanism.

1.3. Authentication, Authorization and Roles

Security in ETICS is based on users and roles. Authorisation and authentication is achieved using x509 client digital certificates.

By default all read-only operations like browsing projects, executing a checkout command and performing local builds is open to any user, even users not registered with the ETICS system. However, although the execution of VCS commands is open to anonymous access, the actual result of the operation depends on the VCS system security, which is independent from ETICS. For example, if anonymous access is disabled in CVS, the user will have to provide authentication, something that is outside the scope of ETICS¹.

Operations that require altering the information in the system or using system resources, like editing modules and configurations or submitting remote builds and tests, can only be performed by registered users with the appropriate role.

Users are identified by the x509 Distinguished Name (DN) that appears in a client certificate issued by a recognised Certification Authority. ETICS recognises as valid the CAs that are part of the EUGridPMA distributions.

In addition to being registered as users in the system, users must have one or more roles in order to perform operations. The following roles are defined:

¹ A solution to secure accesses to VCS systems is for users to deploy private machine and attach them to an ETICS resource pool. This topic is outside the scope of this manual. For more information on this, please contact the ETICS Support team at et:ics-support@cern.ch.

Table 10: Roles

Role name	Allowed operations
Administrator	All operations on all projects. This is a role used only by the ETICS service managers
Module Administrator	Can edit modules and manage security
Developer	Can edit configurations and submit remote builds
Integrator	Can edit configurations, submit remote builds and register build artefacts in the repository
Tester	Can submit remote tests and register test artefacts in the repository
Release Manager	Can lock configurations, submit remote builds and register artefacts in the repository
Guest	Can only perform read-only operations

Roles apply to specific objects in the system and are inherited by objects in the same tree from top to bottom. For example a user can have the role of *Module Administrator* for a project, which gives him or her the same role on all subsystems and components in that project.

Authentication and authorization can only occur when the ETICS system is accessed using the secure protocol 'https'. When http is used, only Guest access is possible.

2. ETICS PORTAL

2.1. Overview

The ETICS portal provides a web-based interface to all ETICS functionality. It is the main web entry point to the ETICS system. It offers an easy way to navigate between different system domains, offers common look and behaviour to facilitate the user experience.

The portal is publicly available at this location: <https://etics.cern.ch/eticsPortal>.

The application supports two main browsers: Firefox (version 2.x or higher) and Internet Explorer (version 6.x or higher). User authentication is based on x.509 user certificates installed in the browser.

The portal uses asynchronous client-server communication mechanism where only a part of a page is refreshed when new data arrive from the server. This improves significantly the application response time and lowers network bandwidth usage. On the other hand, due to asynchronous communication users **cannot bookmark** in their browser a given portal page or interaction state – after opening any bookmarked portal session, the portal will always restart from the default page. For the same reasons using the browser's **refresh button is not recommended** – since it will restart the portal session.

2.2. Layout

The portal has been divided into **panels** that correspond to different system areas – they are organised as tabs accessible at the top of the portal page. When selecting a panel, its content is maximised in the window. Any previously selected panel and any modification made to them are preserved and available when the user selects one of these panels again.

Additionally, every main panel can be divided into **sub-panels** available via smaller tabs at the top of the selected panel – they work exactly in the same manner as the main panels.

Here's a screen shot of the portal and a brief layout explanation:



Figure 4: Portal layout

The portal adapts itself to the user, where it shows only the panels and sub-panels that are appropriate for the current user according to his/her privileges.

The next sub-sections describe one-by-one the different panels in the portal.

2.3. Panel – MyETICS

The MyETICS panel is composed of sub-panels, and is the default panel of the portal.

2.3.1. Welcome

The *Welcome* sub-panel shows the following information:

- The ETICS system, the portal, support channel, user manual.
- The user – his/her identity in the system.
- Available panels.

This panel is available for every user.

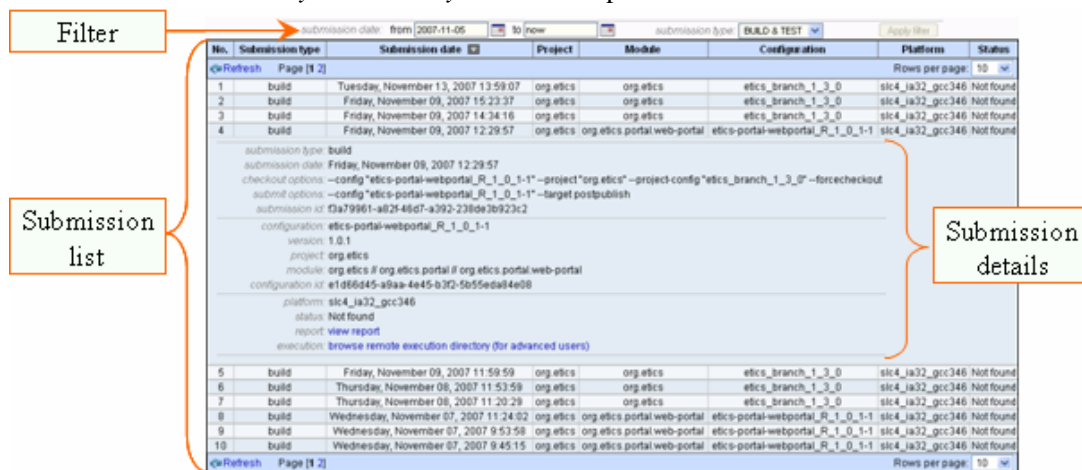
2.3.2. My submissions

The *My submission* panel shows job submitted by the current portal user. This panel is available for registered and active users only.

Submissions are sorted by date starting from the most recent one; the list is divided into pages. After clicking on a row, the user can see the **submission details**, such as the checkout parameters.

A **filter** on the top of the panel allows filtering submissions by given date range and a submission type. The button “*apply filter*” become enabled after the filter is modified. By default, the filter is set to show all the current user submissions from the last 7 days.

Here’s a screenshot of the *MyETICS* -> *My submissions* panel:



Filter

submission date: from 2007-11-05 to now submission type: BUILD & TEST Apply filter

Submission list

No.	Submission type	Submission date	Project	Module	Configuration	Platform	Status
1	build	Tuesday, November 13, 2007 13:59:07	org.etics	org.etics	etics_branch_1_3_0	slc4_ja32_gcc346	Not found
2	build	Friday, November 09, 2007 15:23:37	org.etics	org.etics	etics_branch_1_3_0	slc4_ja32_gcc346	Not found
3	build	Friday, November 09, 2007 14:34:16	org.etics	org.etics	etics_branch_1_3_0	slc4_ja32_gcc346	Not found
4	build	Friday, November 09, 2007 12:29:57	org.etics	org.etics.portal.web-portal	etics-portal-webportal_R_1_0_1-1	slc4_ja32_gcc346	Not found

Submission details

submission type: build
 submission date: Friday, November 09, 2007 12:29:57
 checkout options: --config "etics-portal-webportal_R_1_0_1-1" --project "org.etics" --project config "etics_branch_1_3_0" --forcecheckout
 submit options: --config "etics-portal-webportal_R_1_0_1-1" --target postpublish
 submission id: f3a79861-a82f-48d7-a392-238de3b923c2
 configuration: etics-portal-webportal_R_1_0_1-1
 version: 1.0.1
 project: org.etics
 module: org.etics if org.etics.portal if org.etics.portal.web-portal
 configuration id: e1d66d45-a8aa-4e45-b3c2-5b55eda04e08
 platform: slc4_ja32_gcc346
 status: Not found
 report: view report
 execution: browse remote execution directory (for advanced users)

Figure 5: MyETICS -> My submission layout

2.4. Panel – Build and Test system

The *Build system* panel embeds the ETICS web application described below in chapter 3 (“The ETICS Build and test”).

This panel is available for every user.

2.5. Panel – Repository

The *Repository* panel embeds the ETICS Repository Web Client described in more details in chapter 13 (“Repository”).

This panel is available for every user.

2.6. Panel – administration

The *Administration* panel includes the ETICS administration application described in more details in chapter 15.1 (“The ETICS Administration Application”).

This panel is available only for **system** and **module administrators**.

2.7. Panel – externals

The *Externals* panel allows user to request the creation of a new components and/or a new configurations in the ETICS *externals* project (display name “*All external components*”).

It is available for all registered and active users.

This application is implemented as a wizard. Users can manoeuvre between the wizard steps using two buttons (*back/next*) located at the top and the bottom of the page. The *next* button is available only if all the data in the current wizard step is correctly filled.

The request form consists of two main parts. Firstly, the user must either select from a tree (and optionally modify) an existing component, or create a new component by providing the following information:

1. Component name – to uniquely identify the component in the ETICS system.
2. Component description.
3. Component vendor.
4. Component license type.

Secondly, the user must provide details about the component configuration which corresponds to a particular component version:

1. Configuration name – to uniquely identify the component in the ETICS system.
2. Description.
3. Version.
4. Platform – one of the ETICS platforms.
5. Binary location – where the ETICS team can found the configuration binaries.
6. Reason – a justification why the new component and configuration are needed.

When these two steps are completed, the request is sent to the ETICS team for further processing. The user is then notified by e-mail after his/her request has been accepted or rejected by the ETICS support staff on-duty.

Here is a screenshot of the *Wizard* to request a new external component:

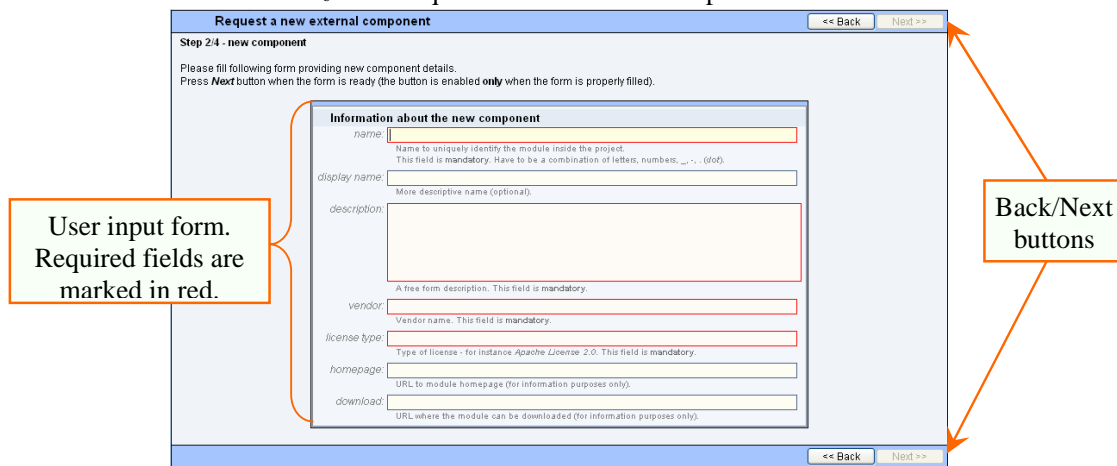


Figure 6: Externals request wizard layout

2.8. Panel – Process new external component request

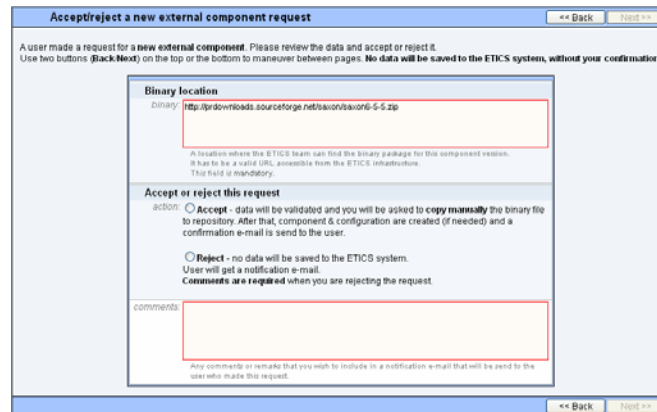
The *Process a new external component request* panel allows the ETICS system administrator to accept or reject a request for adding a new component in the ETICS external area (see previous chapter).

This panel is available only for **system administrator** after he/she opens a link from an e-mail that is send automatically by the system after the request is triggered by the user using the *Externals request wizard*. Additionally, in order to accept the request, the person accepting the request must have **write access** to **CERN AFS** repository¹.

This application is implemented as a wizard and is very similar to requesting a new external component (see previous section). The administrator checks the submitted information and corrects it if needed. If the administrator accepts the request, the wizard instructs him/her to **execute manually** a UNIX command on a machine having access to the infrastructure able to physically copy the component binaries to the ETICS repository. The operation will succeed only if the administrator has write access to the CERN AFS area where the ETICS repository is located.

At the end, a notification e-mail is sent to a user that made the request.

The following screenshot shows an example of the Wizard to process a new external component request:



Accept/reject a new external component request

A user made a request for a new external component. Please review the data and accept or reject it.
Use two buttons: **Back** **Next** on the top or the bottom to maneuver between pages. **No data will be saved to the ETICS system, without your confirmation.**

Binary location

Binary:

A location where the ETICS team can find the binary package for this component version.
It has to be a valid URL, accessible from the ETICS infrastructure.
This field is mandatory.

Accept or reject this request

action: ☒ **Accept** - data will be validated and you will be asked to **copy manually** the binary file to repository. After that, component & configuration are created (if needed) and a confirmation e-mail is send to the user.

☐ **Reject** - no data will be saved to the ETICS system.
User will get a notification e-mail.
Comments are required when you are rejecting the request.

comments:

Any comments or remarks that you wish to include in a notification e-mail that will be send to the user who made this request.

Back Next

Figure 7: Processing of externals request wizard layout

This panel is available for every user.

¹ This requirement access might eventually disappear if the new repository completely replaces the AFS-base repository.

3. THE ETICS BUILD AND TEST WEB APPLICATION

3.1. Overview

The Build and Test Web Application (WA hereafter), together with the ETICS command-based tools, provides user access to build and test capabilities of the ETICS system. Besides being a tool for inspecting and modelling projects within the ETICS infrastructure, the WA also enables easy remote build and test submission and access to build and test reports and artefact.

Access to the ETICS WA is granted to everybody without any authentication/authorization restriction. By default, and this can be modified for each ETICS service installation, anonymous read-only access is granted to all users.

On the other hand, editing grants can be autonomously managed in a fine-grained fashion within any hosted project letting project managers the full control over authorization aspect. For authentication, X509 certificates are used to identify users.

Several deployments of the ETICS system, including WA are currently available¹. The main WA installation is reachable via the ETICS Portal at the following address, selecting the “Build system” panel:

<https://etics.cern.ch/eticsPortal>

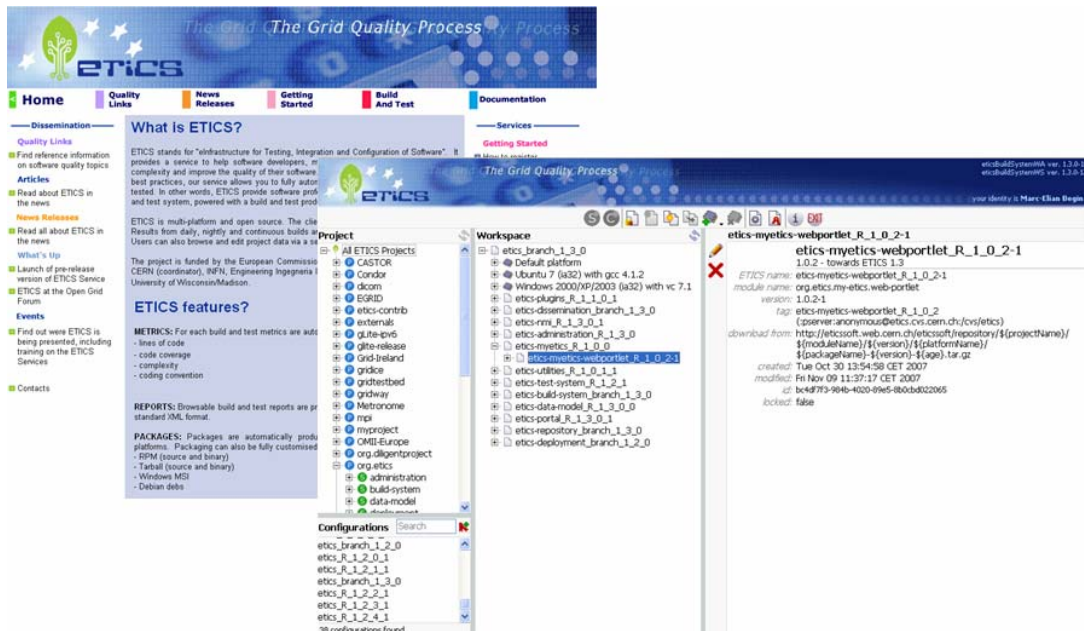


Figure 8 - Build and Test System Web Application

¹ The main ETICS installations are located at CERN, CNAF/INFN and University of Wisconsin-Madison.

3.2. Layout

The current version of the Build and Test System Web Application welcomes the user with a project-selection page. It allows the user to browse, inspect and select the set of projects hosted by the ETICS infrastructure.



Figure 9 - Web Application Project Selector

Once a project is selected, the user is redirected to the main working page where she/he can perform all the main actions concerning the project modelling and build submission. This main page is organised around a four-area layout:

- a browse panel on the top-left side devoted to navigate through the structure of all projects hosted by the ETICS infrastructure.
- a configuration list panel on the bottom-left side with quick search capability
- a workspace panel in middle of the page hosting a number of items¹ the user is currently working on.
- a main edit panel on the top-right side of the page. This area allows either to display metadata for each and every object of the ETICS data model or to host editors for each of them
- a toolbar at the top of the page. This area contains buttons for each action that can be performed on the currently-selected object (i.e. the element shown in the main panel, either in view or edit mode)

The following picture highlights the above-described areas:

¹ In the current release, only one item is allowed in the workspace.

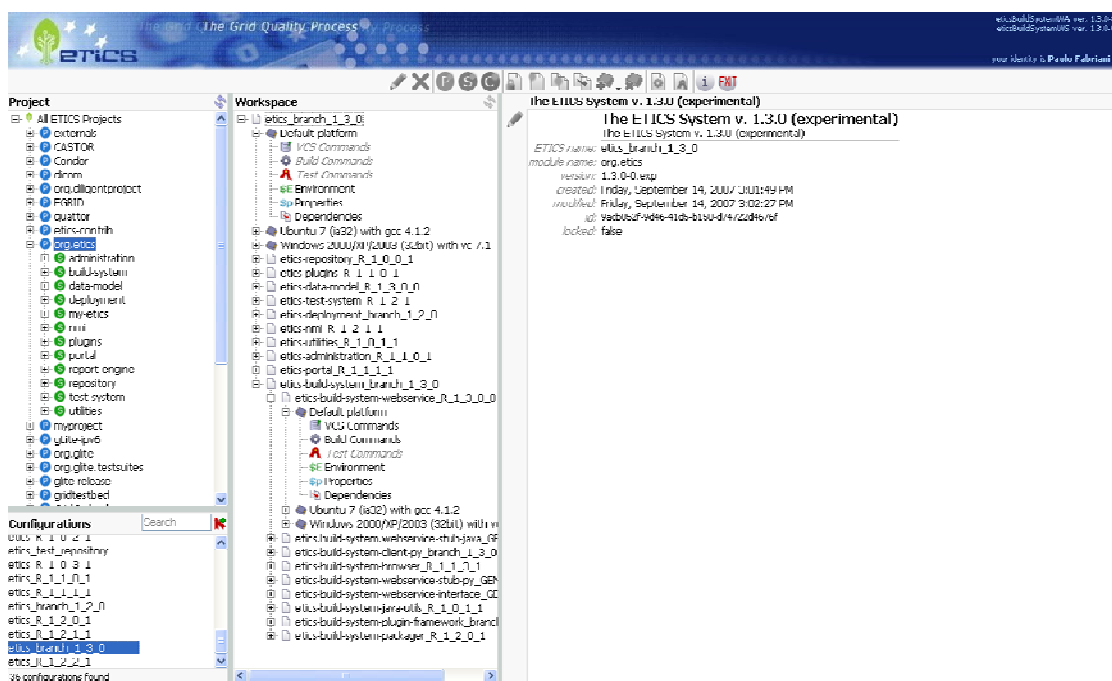


Figure 10 - Build and Test System Web Application Layout

3.3. Enabling Security

In order to be authenticated by the ETICS Service, using the web application, you need to load a valid certificate in your browser. The ETICS production deployment accepts all certificates signed by certificate authorities recognised by the EUGridPMA¹. For test purposes ETICS also has a certificate authority able to sign digital certificates. Users who don't have access to a certification authority recognised by the EUGridPMA can contact the ETICS support team (etics-support@cern.ch) to obtain a recognised certificate.

To load a certificate in the Firefox browser, the certificate must be in the p12 format. The simplest way to load a certificate in Firefox is to:

- On Linux: go to *Edit -> Preferences -> Advanced -> Security*
- On Windows: go to *Tools -> Options -> Advanced*.

Under the *Encryption/Certificates* section, click on 'Manage certificates'. Under the tab 'Your Certificates' click *Import*. Select your p12 format certificate. When prompted, enter the private key password of your certificate.

Once the certificate is loaded, **close and restart Firefox**. To verify that you've loaded the certificate successfully, connect to the ETICS Web Application again, for example:

<https://etics.cern.ch/eticsPortal> (select the panel "Build system").

This time, you should see that you are authenticated with your certificate as indicated in the very top-right corner of the browser window, as shown in Figure 11 below.

¹ <http://www.eugridpma.org/>



Figure 11: Loaded certificate in Firefox (i.e. “Etics User1”)

A similar procedure exists for all browsers supporting digital certificate authentication¹.

The command-line client also uses digital certificates for authentication. For more details on how to use digital certificates for the command-line client, refer to section 4.6 (“Enabling Security”).

3.4. Certificate Registration

Once your certificate has been properly installed in your browser, it can be registered with the ETICS service.

Point your browser to (see Figure 12):

<https://etics.cern.ch/eticsAdmin/public/registration/requestRegistration.jsp>

¹ Please note that at the time of writing, the Build and Test web application only supports Firefox. Support for Internet Explorer and other popular browsers will be added in a future release

Figure 12: Certificate registration form

The request is then validated by the ETICS support team. When your request is accepted, you will receive an email with the activation link. Click on it to **ACTIVATE** your User Account.

4. THE ETICS COMMAND-LINE CLIENT

4.1. Overview

The ETICS Client is a set of command-line tools to access ETICS services. The tools are written in Python and can be installed on any platform where Python ≥ 2.2 .

The installation packages are available from the ETICS repository (see “Chapter 13: Repository” for details) in various formats (e.g., tarballs, RPMS).

The currently recommended way of installing the ETICS Client on most platforms is by using the script:

<http://eticsoft.web.cern.ch/eticsoft/repository/etics-client-setup.py>

This script installs the latest stable version of the client and all required dependencies.

4.2. How to Install the ETICS Client

As mentioned in the previous section, the recommended way of installing the ETICS Client is by using the `etics-client-setup.py` script. This script doesn't require root privileges to be used and only expects Python ≥ 2.2 to be installed.

The following commands can be used to install the client:

On Unix/Linux systems

```
wget http://eticsoft.web.cern.ch/eticsoft/repository/etics-client-setup.py
python etics-client-setup.py
```

The client will be installed in the `'etics'` directory in the directory where the command is executed. An optional parameter `--prefix` can be used to install the client in a different directory.

On Windows systems¹

Download the file from

<http://eticsoft.web.cern.ch/eticsoft/repository/etics-client-setup.py>

and save it in a folder of your choice, then execute:

```
python etics-client-setup.py
```

The client will be installed in the folder where the command is executed. Also in this case the `--prefix` option can be used to redirect the installation to a different folder.

¹ At the time of releasing this document, the client is not supporting Windows

The ETICS Client installation script installes the core client executables and libraries, the test manager and a default set of plugins. It will additionally check for the presence of the following dependencies and install or upgrade them if necessary (the upgrade doesn't affect packages installed outside the ETICS client installation tree):

- 4Suite
- pyXML
- ZSI
- pyOpenSSL
- log4py

OpenSSL is also required and it must be already installed before installing the client.

Finally the installation script checks the presence of and installs if required the ETICS TestManager unit test execution engine.

After installing the client it is recommended to set the following environment variables:

```
ETICS_HOME = <installation_root>
PATH = $ETICS_HOME/bin:$PATH
```

Setting these parameters is required in order to use the same client installation with multiple workspaces. If these variables are not set, the client will only work within the directory where it has been installed (provided the `etics-workspace-setup` command has been run as well, see section 0 for more details on workspaces).

4.3. How to Configure the ETICS Client

The ETICS Client can be configured using the following configuration files:

System configuration file: <installation_root>/etc/etics.conf
User configuration file: <user_home>/.etics.conf
Workspace configuration file: <workspace_root>/etics.conf

The files have exactly the same format and they are used in a specific order. Values in the workspace file have precedence on values in the user file that in turn have precedence on values in the system file. In addition, the workspace file only applies to the current workspace, the user file to all workspaces owned by the current user and the system file to all workspaces of all users sharing the same client installation.

Some parameters can also be set using environment variables (see table below). In this case the environment variable has lower priority than the workspace file, but higher priority than the user file in order to allow setting per workspace configurations.

The configuration parameters are organized in three categories:

User parameters: parameters to configure how users interact with the ETICS client

System parameters: parameters to configure how the client interacts with the ETICS services

Properties: parameters to modify the behaviour of some client commands or how builds and tests are executed

The ETICS client is installed with a preconfigured system configuration file with valid default values for read-only operations. Users can copy the system file and use it as template for the user and workspace files. Note that in the case of the user file, the file must be renamed with a leading “.”. The following table shows the currently supported configuration values and their usage:

Table 11: Configuration file parameters

Parameter	Category	Default value	Description
x509_user_cert	user		The x509_user_cert option is used to specify the location of the user public certificate file. This option can also be set by using the X509_USER_CERT environment variable.
x509_user_key	user		The x509_user_key option is used to specify the location of the user private key of the certificate file. This option can also be set by using the X509_USER_KEY environment variable.
vcsroot	user		The value of the version control system root to be used instead of the one specified as default in the ETICS component metadata. It is normally required to set this property in order to have write access to the version control system. This property can contain for example the value of the CVSROOT to be used.
cvsproxy	user		The value of the CVS proxy host to be used to contact the CVS server. If the environment variable CVS_PROXY is set, it has higher priority than the values set in the system and user configuration files, but lower than the workspace configuration file
updateNotification	user	True	If this parameter is True or missing, a notification message is displayed when running any etics commands when a new version of the client is available. Set this to False not to display the message
protocol	system	https	The protocol used to connect to the ETICS service endpoint. Valid values are [http https] If this option is not set, the default value is https (secure connection)
server	system	etics.cern.ch	The fully qualified hostname where the ETICS service is running
port	system	8443	The IP port used to connect to the ETICS service

updateURL	system	http://eticsoft.web.cern.ch/eticsoft/repository/etics-client-setup.py	<p>If this option is not set, the default values are 8080 if the http protocol is used 8443 if the https protocol is used</p> <p>The default URL of the etics-client-setup script, used to check for new versions. If this parameter is missing the default value displayed on the left is used</p>
platform	properties	Automatically detected	<p>The platform name to be used for checkout, build and test operations. If this property is set here, this value overrides the automatic platform detection. It can be used in case the local platform is not a supported ETICS platform, but compatible with one of the supported platforms</p>
repository	properties	<workspace_root>/repository	<p>The path of the local repository cache. Normally the repository cache is local to each workspace, but the location can be overridden here for example to redirect the cache to another location or to share the same cache among multiple workspaces</p>

4.4. Workspaces

A workspace is a directory in your system where all build and test operations are performed. It is possible to create many workspaces in the system and use them to build different projects or different configurations of the same project. The different workspaces do not interact with each other and removing a workspace removes any trace of what was built or tested in that workspace.

When code is checked-out (prior to a build and/or test), all source code from version control systems (like CVS), as well as all source or binary package required as dependency are downloaded from the repositories into a local cache inside the workspace called *repository*, thus making the workspace self-contained and minimising the system requirements to perform builds and tests.

In order to preserve disk space, it is possible to share the local repository cache among different workspace by using the 'repository' configuration parameter in the ETICS Client configuration files.

Workspaces are created simply by creating a directory of your choice and running the `etics-workspace-setup` command in it. The command will configure the workspace for first use.

4.5. ETICS Commands and Libraries

The ETICS Client is composed of a number of commands and libraries. The commands are found in

```
<installation-root>/bin
```

On Linux systems they do not have the extension 'py', while they keep the extension on Windows. The libraries are found in

```
<installation-root>/lib/pythonX.Y/site-packages (Linux)
```


`<installation-root>/lib/pythonXY (Windows)`

where X and Y represent the installed version of python, for example 'python2.2'.

By convention all commands have a `--help` and a `--version` options that can be used to get more information about the command and the version of the client. The command

`et:ics-version`

can be also used to get the client version and other information (copyright, etc)

4.6. Enabling Security

In order to be authenticated by the ETICS Service, using the command-line client, you need to install a valid digital certificate on your local machine. The ETICS production deployment accepts all certificates signed by certificate authorities recognised by the EUGridPMA¹. For test purposes, ETICS also has a certificate authority able to sign digital certificates. Therefore, users who don't have access to a certification authority recognised by the EUGridPMA, can contact the ETICS support team (et:ics-support@cern.ch).

To instruct the client to use a certificate, copy your certificate in PEM format on your local machine and configure following parameters in the client config file:

- `x509_user_cert` with the location of the public certificate file
- `x509_user_key` with the location of the private certificate file

The first time you use an ETICS command, you will be prompted to enter the passphrase of your private key. Once this is done, your private key will be encrypted and stored in a secure way such that you do not have to re-enter your passphrase every time you use the client.

Here's a recipe² to convert a digital certificate from `.p12` to `.pem` format on Linux:

Execute the following command (you will be prompted for a password):

```
> openssl pkcs12 -in cert.p12 -clcerts -out cert.pem
```

where `cert.p12` is a certificate in `.p12` format (i.e. PKCS12) and `cert.pem` is the output certificate in `.pem` format.

Users can also define the following environment variables instead of the config file: `X509_USER_KEY` and `X509_USER_CERT`. Note that if both are set (environment variable and config file), the values from the environment variables will be taken.

¹ <http://www.eugridpma.org/>

² Recipe based on the following document: <http://www.openssl.org/docs/apps/pkcs12.html>

5. BROWSING MODULES AND CONFIGURATION INFORMATION

5.1. Overview

This chapter describes how to retrieve and browse information about project, subsystems, component, configuration and all related objects using the ETICS Web Application and the ETICS Client. All methods described in this chapter can be used by anonymous users and do not require configuring the web browser or the client for secure access. However, if security is configured, also the read-only command will use it and will ask for the certificate passphrase if necessary.

5.2. How to Browse with the Web Application

In this section, you will learn how to view and browse metadata using the web application.

5.2.1. Selecting a Project

For most functionality, the usage of the Web Application (WA in this section) is scoped to a specific project. For that, the welcome page of the WA provides the user with the list of the projects currently managed by the ETICS infrastructure.

Projects can be selected using the drop-down list in the top part of the project selector. After selecting one, details of the project are displayed just below. Once the desired project is found, the arrow-shaped button on the right-bottom gives you access to this project.

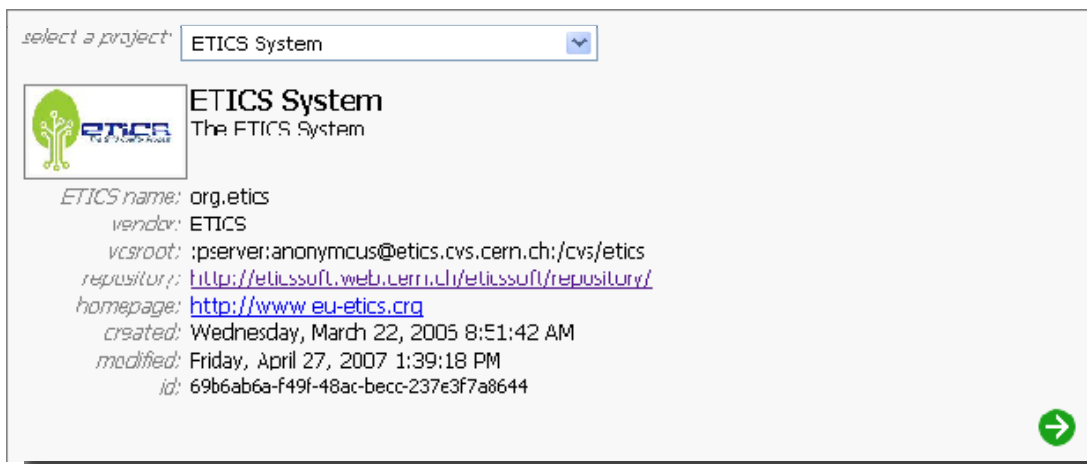


Figure 13: Project selection form

5.2.2. Browsing a Project, Subsystems and Components

Once a project has been selected, the main page of the WA shows the project structure on the left, project details on the right and project configurations in the configuration list, whereas the workspace is initially empty.

On the left-hand side of the page, a tree-like component shows the structure of the current project; expanding the root, subsystems and components are progressively shown¹. An icon beside the module name² tells what type the module is (i.e. project, subsystem or component).

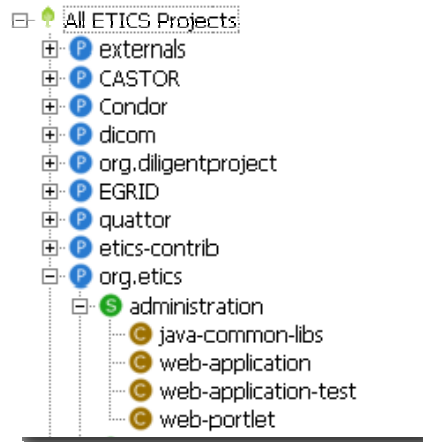


Figure 14: Module tree

A single click on a node in the tree triggers an update of the main panel on the right side of the page, as well as an update of the list of configurations available for the selected module. In the main panel, the details area is titled with the module *displayName*³ and *description*. Then, all module metadata is provided with appropriate formatting. In the configuration list, the available configurations for the selected module are displayed, sorted by last modification date.

¹ A lightweight approach has been adopted for the tree component. After clicking a node the first time, child modules are not immediately available as a request for them is being sent to the ETICS server.

² If fully qualified names are adopted, for clarity, just the most significant part of the name with respect to the parent name is displayed in the tree. The full name of the component is shown in the module details area.

³ If no *displayName* attribute is set for the module, the *name* attribute is used to title the area. In this case, the title is italicised

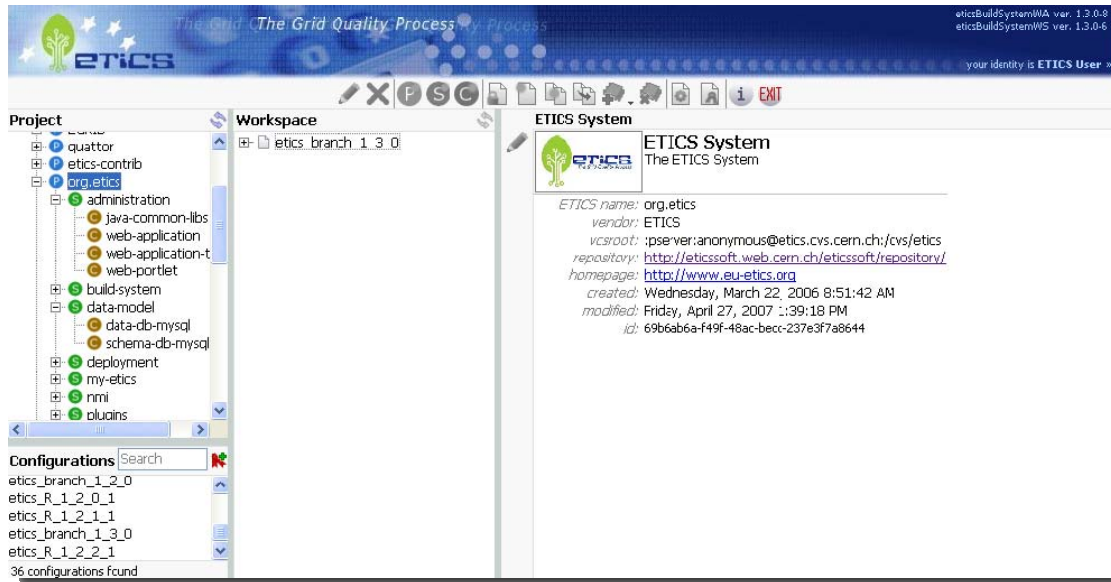


Figure 15: Module details

It may happen that, while browsing a project, the corresponding data-model changes on the ETICS System. In this case, because of the proxying policy adopted in the WA, the information available could be out-to-date. To synchronize the tree cache with respect to the latest project data-model, perform a refresh of the node. This can be done by clicking the 'refresh' button on the top-right corner of the panel, as well as using a drop-down menu on the tree node.

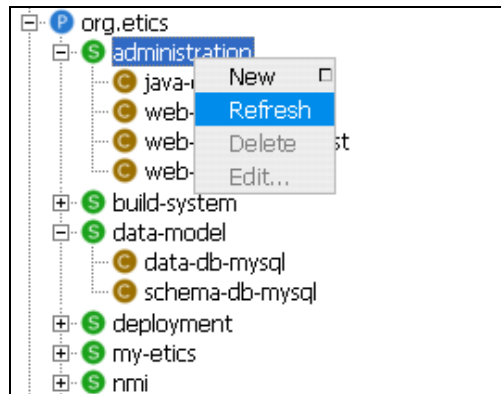


Figure 16: Refresh a module

5.2.3. Searching Configurations

Upon selection of modules in the project tree, the configuration list panel is automatically populated with all the available configurations for the selected project. The list of configurations is initially sorted according to the last-modification date.

By selecting items in the configuration list, configuration's metadata is displayed in the main panel on the right-side of the page.

When the number of configurations for a module becomes large, finding a particular configuration can become difficult. To ease this job, a Search can be performed over the set of module configurations.

The search action can be activated by specifying a keyword in the search field and pressing the ‘Enter’ key¹. The configuration list is then updated with the result set, whose size is displayed in the status bar of the list.

Search results are not persisted, i.e. after selecting a different module in the tree, the content of the list is discarded and updated with a new list.

5.2.4. Adding configurations to the Workspace

Configurations can be added to the workspace by selecting them and then clicking the ‘bookmark’ button on the top-right side of the panel, or by using the contextual menu on the configuration. In the current release, the workspace can host only one configuration; thus adding a configuration results in the replacement of the current configuration (if any) in the workspace.

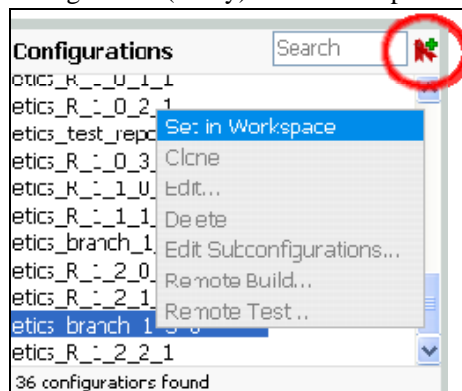


Figure 17: Selecting configuration

5.2.5. Browsing Configurations and Sub-Configurations

As with modules, to inspect the configuration’s metadata, first select a configuration. Currently this can be done in two areas of the WA: the *configuration list* and the *workspace*.

The *configuration list* is located in the bottom-left part of the page and lists all the configurations associated with a selected module. By clicking one of them, details of the configuration are displayed in the *details area*.

¹ In the current release, the match algorithm simply looks for the search key within the configuration name.

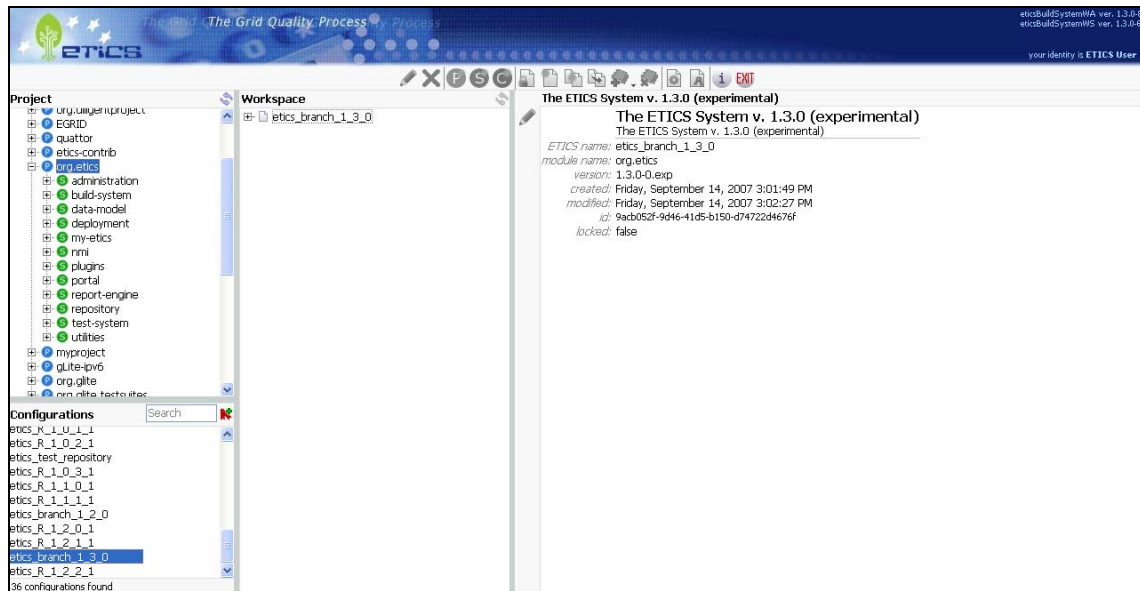


Figure 18: Listing configurations

Similar to modules, configurations are hierarchical structures constrained by the associated project structure. This structure can be browsed by using the *workspace* available in the central area of the page. Initially, the workspace is empty; refer to the previous section to see how to change its content.

Expanding nodes in the workspace, sub-configurations are shown as children¹. Clicking on them, details are provided in the right-hand side of the page.

When expanding configurations and sub-configurations, *platform* nodes, if defined, are displayed; details on this kind of nodes will be given in the following section.

It may happen that, while browsing a configuration tree, the corresponding data-model changes on the ETICS System. In this case, because of the proxying policy adopted in the WA, the information available could be out-of sync. To synchronise the tree cache with respect to the latest project data-model, perform a *refresh* of the node, by clicking the 'refresh' button on the top-right corner of the workspace or by using the drop-down menu on the tree node.

5.2.6. Listing supported Platforms

The expansion of a configuration node, besides revealing the configuration structure, often shows a number of 'platform' nodes; details of each of them can be displayed, again, by clicking on the node.

¹ If fully qualified names are adopted, for clarity, just the most significant part of the name with respect to the parent name is displayed in the tree. The full name of the configuration is shown in the configuration details area.

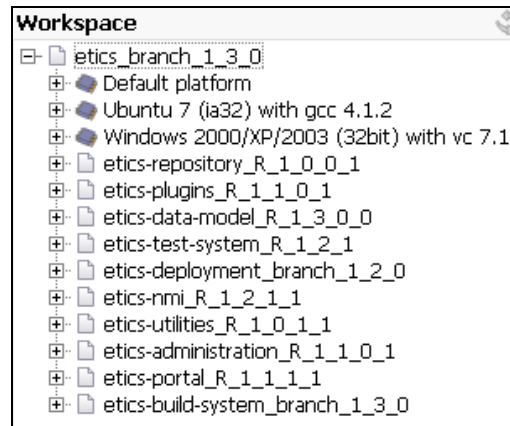


Figure 19: Supported Platforms

Different specific platforms can be attached to a configuration, including the '*Default Platform*' providing commands, properties, environments variables and dependencies to be used when no specific platform is defined.

5.2.7. Viewing Commands, Properties and Dependencies

Expanding a platform node, platform-dependent items are displayed if defined for the corresponding platform. Namely VCS commands, Build Commands, Test Commands, Properties, Environment variables and Dependencies. As usual, details of any selected item are shown on the right side of the page. When a set of commands (i.e. Build, Test, VCS) is not defined for a configuration and a platform, the label of the node is rendered in italic style and gray colour.

Commands, properties and environment variables viewers simply show attribute-value or key-value pairs. The Dependency viewer shows the list of required software for the current configuration. In addition, both the type (static and/or dynamic) and the scope (build and/or runtime) of each dependency are displayed beside each dependency entry.

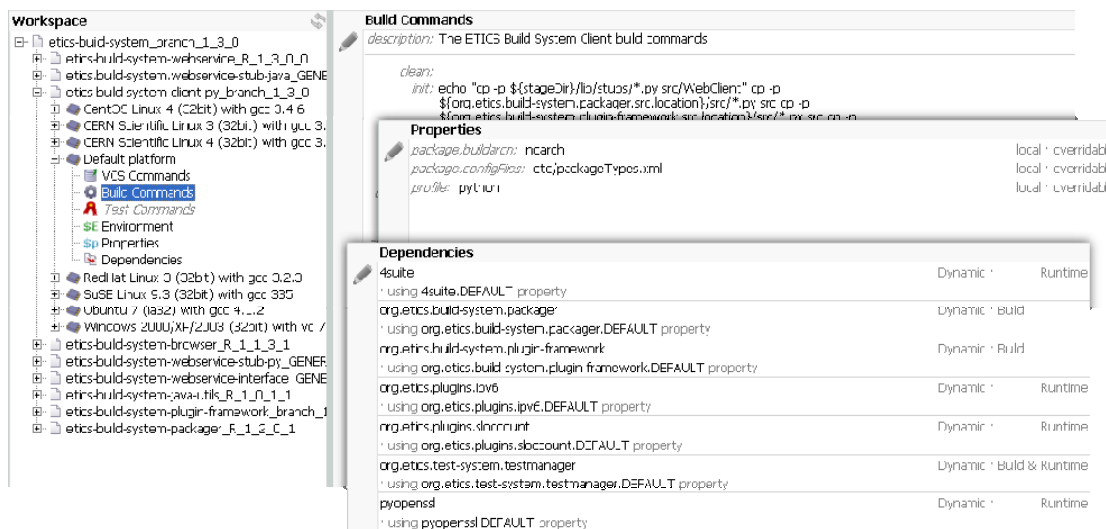


Figure 20: Viewing configuration details

5.3. How to Browse with the Command-Line Client

In this section, you will learn how to view and browse metadata using the command-line client.

The following actions assume that the client has been correctly installed and configured and that the current workspace has been configured using the `etics-workspace-setup` command (see section 0).

5.3.1. How to Get a Project

Before any action can be taken in a project, the project information (metadata) must be extracted from the ETICS metadata store into the workspace.

The action of getting a project into the workspace can be compared to the action of setting the `CVSROOT` for CVS. It gives the context where all subsequent operations are performed. The command to get a project is:

```
etcs-get-project <project-name>
```

If you are not sure of the project name or just want to know what's available the following command can be used:

```
etcs-list-project [options] [<project-name>]
```

This command by itself prints a list of the available projects. If the `-d` option is used, the properties of each project are also printed. An optional project name argument can be passed to get information only about a specific project.

All information retrieved by `etcs-get-project` and other commands are cached locally in the *store.xml* file in at the root of the workspace. It is not recommended to manually edit this file, since the information may get out of synch with the ETICS server data-store.

5.3.2. How to Checkout and Browse Configurations

Once a project has been inserted into the workspace, all information about the project structure (its subsystems and components) is available locally. However this information is not enough to perform builds or tests, since additional configuration information and the actual source code or binary packages are required. In order to get the configuration information, source code and packages downloaded and installed onto the workspace, the following command is used:

```
etcs-checkout [options] [<module-name>]
```

The `etcs-checkout` command has many options and its behaviour is described in details in the following chapter. For the moment we will look at a few simple cases. It is useful to note that the syntax of the `etcs-checkout` command is similar to the syntax of the `cvcs checkout` command.

Used by itself, the `etcs-checkout` command gets the HEAD configuration of the current project and all attached subsystems and components configurations and dependencies for the current working platform. More information about configurations is given in the following chapter (Chapter 6 - Checking out Project, Subsystems and Components), but in general we can assume that all modules in

the system have a HEAD configuration. The HEAD configuration is created, with each newly created module, with default values. The default HEAD configuration name is by convention:

```
module-name.HEAD
```

If the <module-name> argument is used, it is possible to check out the HEAD configuration of a specific module (subsystem or component) in the project tree:

```
etics-checkout <module-name>
```

In order to download a specific configuration from the system, the configuration name must be specified, just like you have to specify a tag name when checking out code from a CVS tag or branch:

```
etics-checkout -c <config-name> <module-name>
```

or

```
etics-checkout --config <config-name> <module-name>
```

If you are not sure of the configuration name of a module or just want to know what's available the following command can be used:

```
etics-list-configuration [options] [<module-name>]
```

This command by itself prints a list of the available configurations for the current project. If the -d option is used the properties of each configuration are also printed. If the optional module name argument is passed, the command lists the configurations of that module.

Additionally, the command

```
etics-list-property [options][<module-name>]
```

can be used to get a list of the properties of given module. If no module is provided as argument, the command prints the properties of the current working project.

5.3.3. How to Show the Structure of Modules and Configurations?

The command:

```
etics-show-configuration-structure [options] [<module-name>]
```

can be used to get a visualise of the structure of a configuration. The command displays on the screen a list of all children and dependencies of the current configuration of the specified module in the order they would be built. As for the previous commands, the command `etics-show-configuration-structure` by itself prints the structure of the HEAD configuration of the

current project. A number of options are available to change the way the information is displayed (as a flat list instead of an indented hierarchical list) or to output an xml file instead of a text list.

This command only work with local metadata therefore requires an `etics-checkout` to have been performed before.

```
etics-show-module-structure [options] [<module-name>]
```

accepts as argument the name of a module (could be a project, a subsystem or a component) and prints on the screen the module hierarchy. If no module is specified the command shows the structure of the current project. In order to get the structure of a component or a subsystem the corresponding option must be used (`--component` if it's a component, `--subsystem` or `-s` if it's a subsystem). The `-d` option is available to display more details.

5.3.4. Modules, Platforms, Users, Roles and Other Objects

The ETICS command line client provides a set of tools to list and get information about several of the different objects in the ETICS metadata store. Some of these tools have been described earlier (`etics-list-project`, `etics-list-configuration`). The following additional commands are also available:

```
etics-list-platform [option] [<platform-name>]
```

can be used to get a list of the platforms supported by the ETICS system or get the details of a specific platform. The `-d` option can be used to get the platform properties. The `--detect` option displays the platform string corresponding to the local platform, which may or may not exist in the ETICS metadata store.

```
etics-list-user [option]
```

by default returns the list of users registered in the ETICS system. If `-d` option is used, it returns also the projects for which the users are registered and the roles they have on it.

The `--user (-u) <user-last-name>` option can be used to get information about a specific user. With this option, the command lists the user DN (as defined in the user's certificate) and the e-mail address. In addition, if the `-d` option is also given, the modules for which the user is registered and the related Roles are printed on the screen.

Finally the `--project (-p) <project-name>` option can be used to return the list of users registered for a given project. If `-d` option is also used, more detailed information is printed out about the users and roles for each module within the project.

The `--user` and `--project` options cannot be used together.

6. CHECKING OUT PROJECT, SUBSYSTEMS AND COMPONENTS

This chapter describes how to checkout and download code onto the workspace.

6.1. Overview

This section describes in more details how the `et:ics-checkout` command works to download configuration metadata, source code and binary packages from the ETICS metadata store, as well as source code and package repositories. All operations described in this chapter can be used by anonymous users and do not require configuring the client for secure access. However, if security is configured, the `et:ics-checkout` command prompt the user for the certificate passphrase if necessary.

6.2. Source Code or Binary Packages?

Before going into the details of the `et:ics-checkout` command, it is necessary to understand what type of objects can checked-out.

The ETICS system uses three types of data for software:

1. **Metadata:** the configuration objects in the ETICS metadata store describing how a particular version of a module must be checked out, built or tested
2. **Source code:** the code stored in some VCS (Version Control System) like CVS or Subversion or stored as a tarball in the ETICS repository
3. **Binary packages:** pre-compiled tarballs stored in the ETICS repository.

For more information about the format of the ETICS repository, please refer to Chapter 13 (“Repository”).

The metadata is always required to perform build and test operations. The choice of using source code from a version control system, source code from a tarball in the repository or binary tarballs from the repository depends on user requirements and on the availability of the packages. In the following sections we will explore the various options that affect what type of packages are fetched by the `et:ics-checkout` command.

One more thing to take into account is that both the source and binary tarballs taken from the ETICS repository are cached locally in the `repository` directory (by default there is one such directory in every workspace, but this can be changed by using the `repository` entry in the client configuration files). The repository directory structure is identical to the ETICS central repository structure, but contains expanded copies of the packages required by the current build/test operations.

Conversely, code taken from version control systems is written locally in the workspace according to the instruction set by the `checkout` target in the VCS Commands set of each configuration.

6.3. The `et:ics-checkout` Command

As briefly seen in the previous chapter, the `et:ics-checkout` command is used to extract metadata and code or binary packages from various places. The command is at the base of all build and test operations, since no such operation can be performed without first having the required data locally.

The syntax of the command is:

```
et:ics-checkout [option] [<module-name>]
```

The command `etics-checkout` by itself checks out the metadata and all associated source or binary packages for the HEAD configurations of the current project.

The command has several options that allow controlling how check out is executed. The following table shows the command options:

Table 12: etics-checkout command options

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-c, --config <configuration-name></code>	Define a specific configuration to be used instead of the default <code><module-name>.HEAD</code> . If the <code><module-name></code> argument is defined, the configuration applies to the module, otherwise it applies to the current project.
<code>--project <project-name></code>	Specify a project name to be used instead of the current project (as defined by the <code>etics-get-project</code> command). Use this when checking out components configurations from other projects (for example from the <i>'externals'</i> project). This option is often used with the <code>--merge</code> option.
<code>--project-config <project-configuration-name></code>	Define a specific project configuration for the metadata. If this parameter is specified, the full project metadata is retrieved. This is required when checking out only parts of the project, but there is project-wide information that should be propagated.
<code>-p, --property <property=value></code>	If already define, override existing properties, or define new ones. You can list the available properties and their default value using <i>'etics-list-property'</i> .
<code>--platform <platform-name></code>	Overwrite the local platform.
<code>--nocheckout</code>	Do not perform the actual VCS checkout and/or repository download.
<code>--nodeps</code>	Only checkout the currently specified module (do not checkout children and dependencies)
<code>--shallowbindeps</code>	When checking out using the <code>--frombinary</code> or <code>--frombinaryonly</code> options, dependencies of binary packages are not checked out
<code>--runtimedeps</code>	When checking out configurations, dependencies of run-time dependencies are also processed. This option is useful to make sure that the final list of packages contains everything needed to deploy the components
<code>--local</code>	Do not contact the server to download metadata. This will only work if the configuration information has already being downloaded (via this command, without this option).
<code>--volatile <namespace></code>	Use a named volatile repository to look for packages. If a package is not found in the volatile repository, it is searched for in the permanent repository, before giving up unless <code>--volatileonly</code> is used. This option cannot be used together with <code>--defaultvolatile</code> .
<code>--defaultvolatile</code>	Use the default volatile repository to look for packages. If a package is not found in the volatile repository, it is searched for

	in the permanent repository, before giving up unless <code>--volatileonly</code> is used. This is equivalent to <code>--volatile=default</code> . This option cannot be used together with <code>--volatile</code> .
<code>--volatileonly</code>	Use only the specified volatile repository to look for packages. If a package is not found in the volatile repository, it is not searched for in the permanent repository
<code>--force</code>	Force the configuration checkout or download even if no changes are detected. Note: This option is equivalent to <code>--forcevcscheckout</code> and <code>--forcedownload</code> .
<code>--forcevcscheckout</code>	Forces the checkout of the configuration from VCS, even if the corresponding module hasn't changed with respect to the workspace version.
<code>--forcedownload</code>	Forces the download of the configuration from the repository, even if the corresponding artefact hasn't changed with respect to the repository version
<code>--noask</code>	Doesn't ask user confirmation and assumes YES to all questions
<code>--fromsource</code>	When possible, check out source code instead of downloading binaries.
<code>--frombinary</code>	When possible, download binaries instead of checking out source code. This implies that packages will not be built.
<code>--fromsourceonly</code>	Check out source code only. This means that all the configurations, including dependencies have to be available in source code form or the operation will fail
<code>--frombinaryonly</code>	Checkout binaries only. This means that all the configurations, including dependencies have to be available in binary form.
<code>--continueonerror</code>	Continue when checkout and/or download errors are encountered.
<code>--merge</code>	Merge checkouts with current workspace. Without this option, the metadata stored in the workspace, from previous checkouts, is cleaned-up, while the code checked-out by the VCS commands and downloaded remains in the workspace.
<code>--verbose</code>	Print verbose messages.
<code>--version</code>	Display the client version.

6.4. How to Checkout Source Code or Binary Packages

By default the `et:ics-checkout` command tries to extract from the VCS system or the repository the source code of the module specified as argument for the given configuration. If the module is a project or a subsystem, the command tries to check out the source code of all its children. If the source code is not available, the command tries to get binary packages. All dependencies different from the module or its children are taken in binary format if they exist, otherwise they are taken in source form. The described default behaviour is equivalent to say that the configurations of module specified as argument and its children (if any) are checked out as if the `--fromsource` option had been specified, while all dependencies are checked out as if the `--frombinary` option had been specified.

This behaviour allows checking out and working on the source code of a module during a normal development session, without having to build all the external dependencies.

If one of the options `--fromsource`, `--frombinary`, `--fromsourceonly` or `--frombinaryonly` is specified, it applies to all configurations.

For example:

```
etics-checkout -c etics_R_0_9_2 org.etics.build-system
```

checks out the Build and Test System subsystem of the ETICS system and all its children (e.g. client-py, packager, plugin-framework) in source format from the ETICS CVS repository and checks out the required external dependencies (e.g. ZSI, pyxml, Psycho) in binary format for the local platform.

```
etics-checkout --frombinaryonly -c etics_R_0_9_2 org.etics.build-system
```

checks out the Build and Test System subsystem of the ETICS system and all its children and dependencies in binary format for the local platform. If one or more packages do not exist, the command fails and exits with a non-zero return code.

6.4.1. Merging Configurations

When running the `etics-checkout` command, the local configuration store is normally reset to contain only the new configurations being checked out. However, there may be cases when one or more configurations have to be injected in an existing store, for example to try a different configuration of one component with the existing configurations of other components.

The command:

```
etics-checkout --merge -c <configuration-name> <module-name>
```

performs such a merge operation. As a result of this operation, the new configuration (or configurations in case of subsystems) is added to the store, but it doesn't replace existing configurations of the same module. Therefore it may be necessary to use the `-c` option when building to select one of the existing configurations (see section 9 - Building Configurations for more information).

6.4.2. Updating Configurations

In case an existing configuration has changed in the repository or VCS system and it's necessary to update the local store, a different command can be used. The `etics-update` command performs the same types of operation as `etics-checkout` but updates existing configurations using the same configuration names and version numbers.

```
etics-update <module-name>
```

updates the currently stored configuration of the module <module-name>. The `etics-update` command has the same or equivalent options as `etics-checkout` with the exception of the `-c` option.

6.4.3. Forcing Checkout

The standard behaviour of the `etics-checkout` command is to execute any given action once and then cache some information on disk that prevents executing again the same action if nothing in the code or binary package has changed. This allows speeding up the operation of checking out components if up-to-date packages or code are already available in the workspace.

However in certain occasions it may be necessary to perform the same checkout again, for example because something has changed in a place outside of ETICS's control or the same action has to be repeated with different options. In this case, the build can be force using the `--force` option:

```
etics-checkout --force -c <configuration-name> <module-name>
```

It is also possible to force only the check out of code from a VCS repository without refreshing the packages in the package repository (`--forcevcscheckout`) or to force only the download of packages from the package repository without refreshing code from the VCS system (`--forcedownload`)

6.4.4. Permanent and Volatile Repositories

ETICS provides two types of repositories:

- The permanent repository at <http://eticsoft.web.cern.ch/eticsoft/repository> is used to store packages that can be officially published by a project. The packages are stored in the repository when a remote build is submitted with the option:

```
--register
```

Packages in the permanent repository are never overwritten even if the package is rebuilt.

- The volatile repositories are custom namespaced repositories at <http://eticsoft.web.cern.ch/eticsoft/builds/<namespace>> where namespace is a string supplied by the user. Volatile repositories are used to temporarily store packages during the development phase when they are not yet ready for public general availability and can be used to categorize the builds or provide teamwork support on specific tasks. Packages are store in the volatile repositories when submitting a remote build with the option:

```
--register-volatile <namespace>
```

Packages in the volatile repositories are usually purged every two months (or less depending on available space) and the packages are overwritten if they are built again. All packages built by submitting a remote build without the `--register-volatile` option are registered in the default volatile repository at <http://eticsoft.web.cern.ch/eticsoft/builds/default>

The `etics-checkout` command provides the possibility of checking out code from a volatile repository only or from a combination of a volatile repository and the permanent repository. The command

```
etics-checkout -c <config-name> --volatile=<my-namespace> <module-name>
```

will try to checkout the configuration *config-name* of the module *module-name* from the volatile repository called *<my-namespace>*, if a package cannot be found in the volatile repository, the client looks in the permanent repository. The command:

```
etics-checkout -c <config-name> --volatile=<my-namespace> --volatileonly <module-name>
```

will try to checkout the configuration *config-name* of the module *module-name* from the volatile repository called *<my-namespace>*, if a package cannot be found the client stops.

The shortcut option `--defaultvolatile` can be used to indicate that packages have to be looked for in the default volatile repository <http://eticsoft.web.cern.ch/eticsoft/builds/default>

7. EDITING PROJECTS

7.1. Overview

This chapter describes how to edit information (i.e. metadata) about project, subsystems, component, configuration and all related objects using the ETICS Web Application and the ETICS Client. All methods described in this chapter require the users to authenticate themselves using a digital certificate (see section 3.3 “Enabling Security” for details).

7.2. How to Edit with the Web Application

This chapter describes how to edit project, subsystems, component, configuration and all related objects information (i.e. metadata) using the ETICS Web Application. All methods described in this chapter require the users to authenticate themselves using a digital certificate (see section 3.3 “Enabling Security” for details).

In this section, you will learn how to view and browse metadata using the web application.

All editing functionalities can be activated through the ‘edit’ button available in the main panel for most object types. Alternatively, editable items provide an ‘edit’ item in their contextual menu.

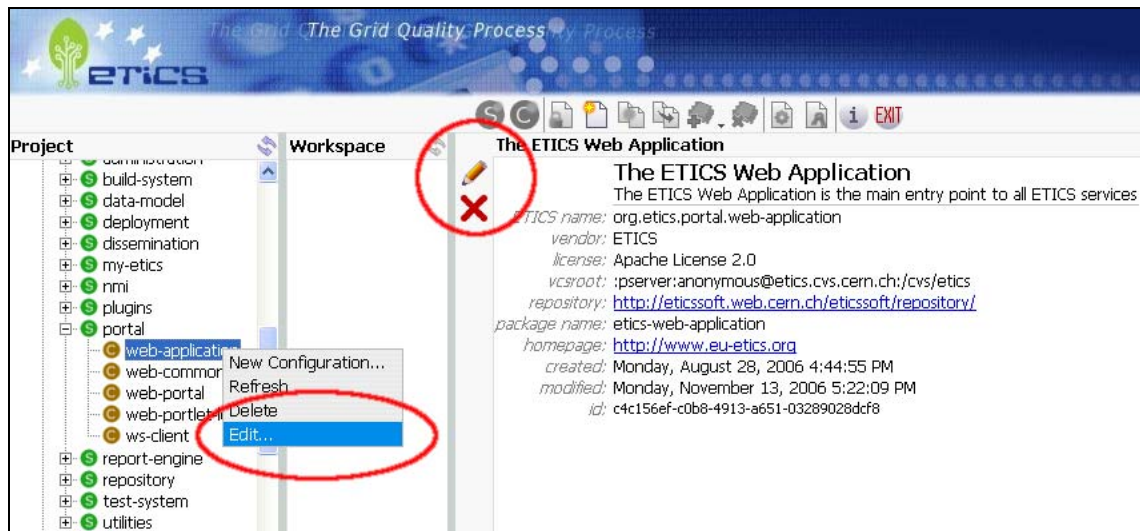


Figure 21 - Editing Etics Objects

7.2.1. Editing Modules

Creating new Modules

According to the ETICS data-model, projects can contain both subsystems and components, whereas subsystems can contain components only. A new subsystem or component can be created using the buttons in the toolbar as well as using the contextual menu on the parent module.

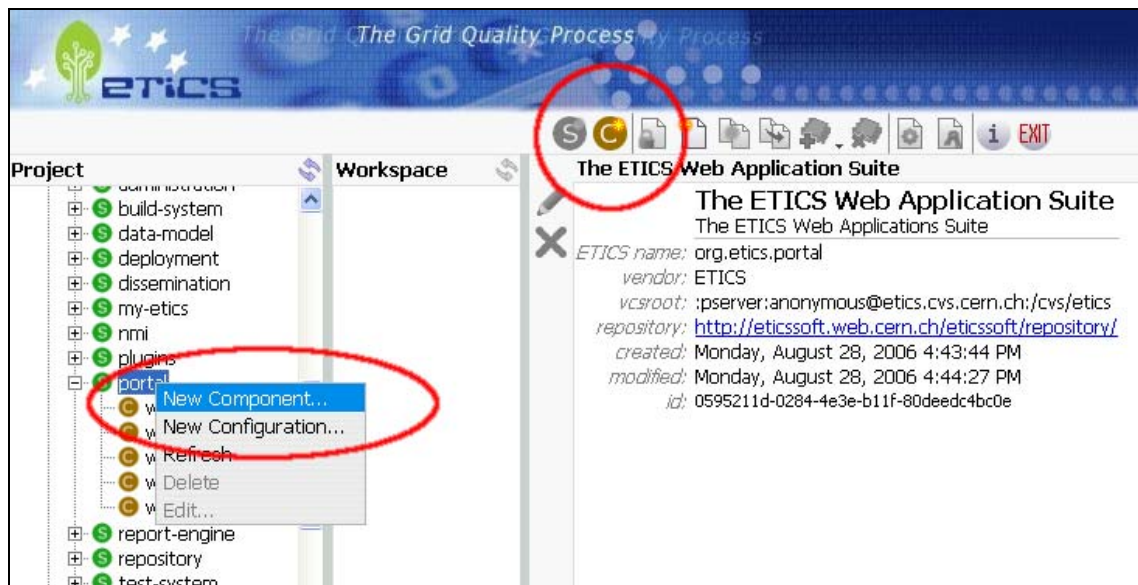


Figure 22: Creating new modules

An empty form will appear in the details area, allowing the user to provide metadata. When finished, click on the 'save' button to store the new module or 'exit' to abort the action.

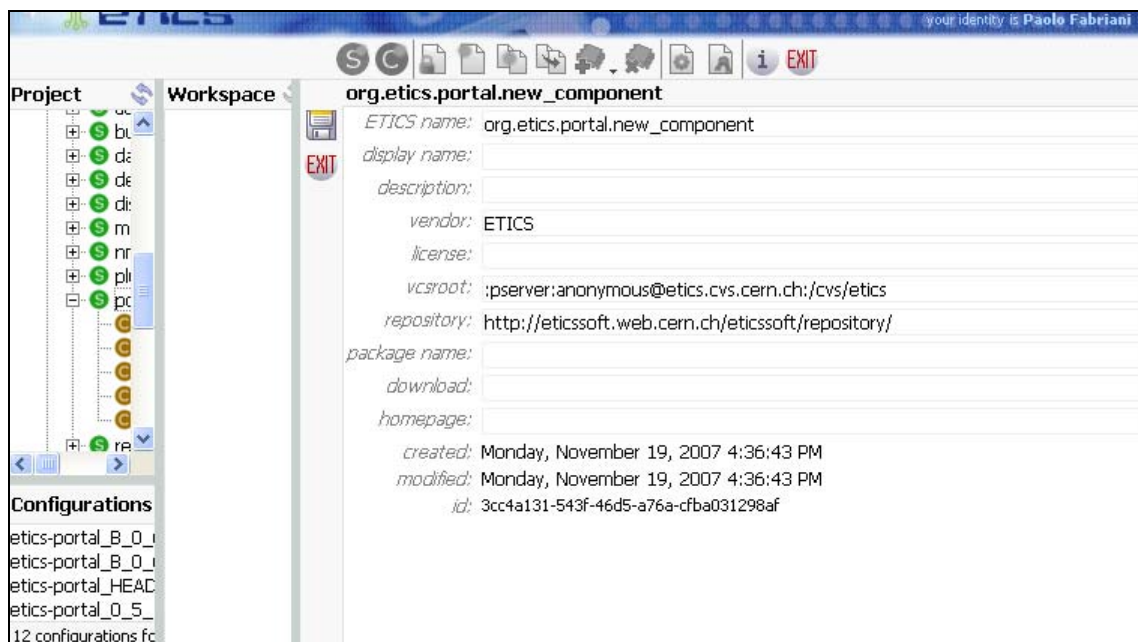


Figure 23: Editing newly created module form

Modifying existing Modules

A module can also be modified after its creation using the 'edit' button or through the corresponding item in the contextual menu.

A form will appear showing the current metadata of the selected module. Most entries can be modified. When finished, click 'save' to update the module metadata or 'exit' to abort the action.

Removing Modules

A module can be removed using the 'delete' button in the main panel or through the corresponding item in the contextual menu of the selected node.

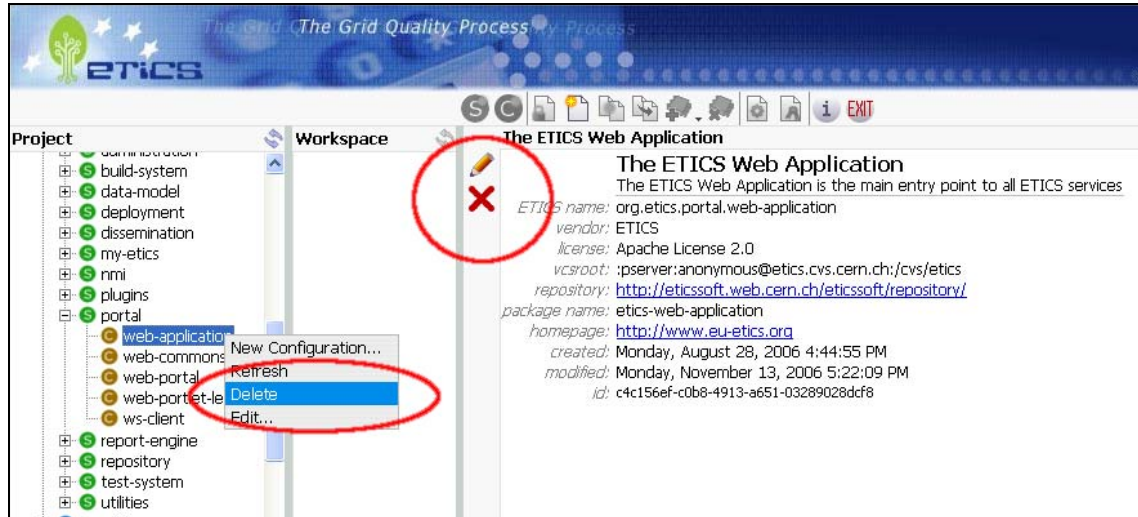


Figure 24: Deleting modules

Note: The removal of a module, also deletes its sub-modules (i.e. subsystems and components), all its configurations and related objects (commands, properties, dependencies, etc.). A project cannot be deleted via the web application. To delete a project, send a request to etics-support@cern.ch.

7.2.2. Editing Configurations

Configuration editing can be activated through the 'edit' button in the main panel or through the popup menu associated with each item in the *workspace* or in the *configuration list*.

Creating new Configurations

A new configuration for a module can be created using the 'new configuration' button in the toolbar or through the contextual menu on the selected module.

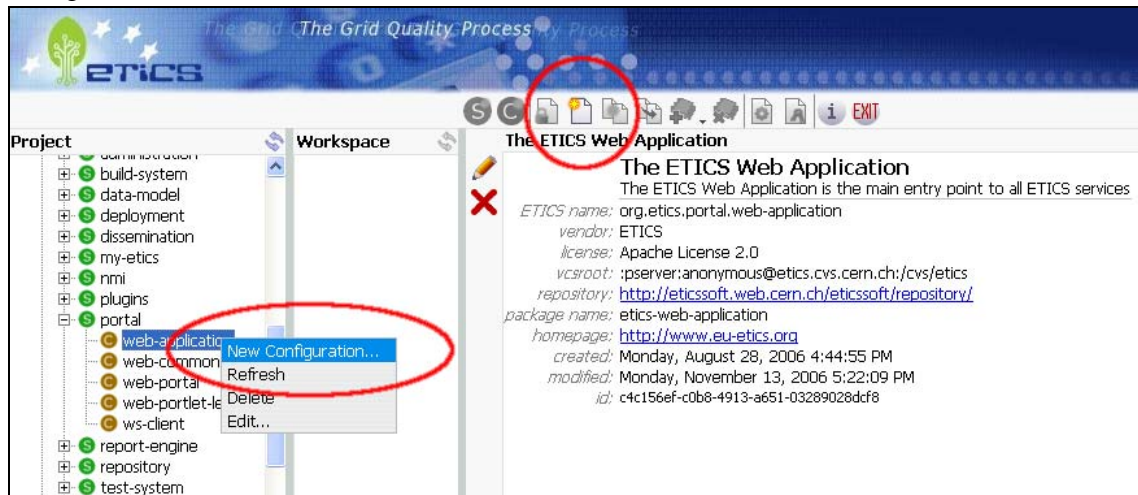


Figure 25: Creating configurations

An empty form will appear in the details area, allowing setting values for user-editable metadata. When finished, click on the 'save' button to store the new module or 'exit' to abort the action.

Modifying existing Configurations

A configuration can be modified using the 'edit' button or through the corresponding item in the contextual menu of the selected configuration.

A form will appear showing the current metadata of the selected configuration. Most entries can be modified. When finished, click 'save' to update the configuration metadata or 'exit' to abort the action.

Removing Configurations

A configuration can be removed using the 'delete' button in the main panel or through the corresponding item in the contextual menu of the selected configuration.

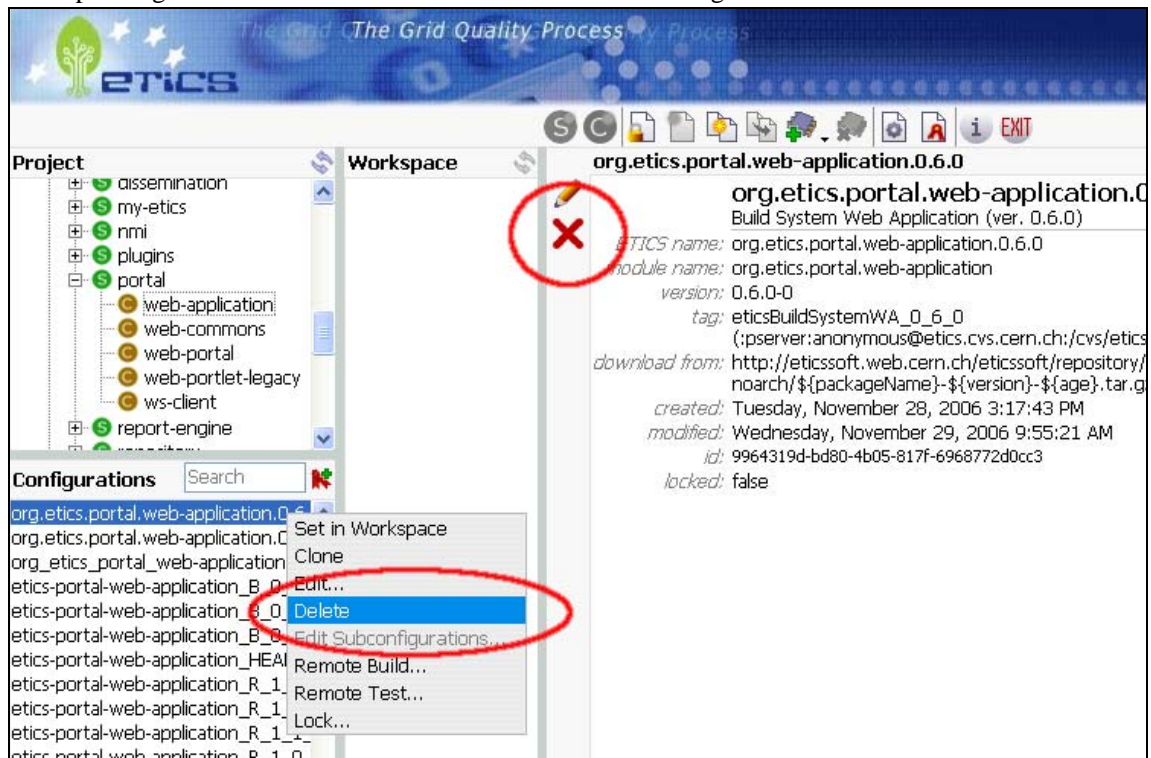


Figure 26: Deleting configurations

Note: The removal of a configuration, also deletes its related objects (e.g. commands, properties, dependencies), and its relationships with its sub-configurations. Further, sub-configurations are not removed and remain attached to the corresponding modules.

Cloning Configurations

The 'Clone' operation creates a new configuration with the very same metadata (with the only exception of the configuration id and name), including commands, properties, environment, sub-configurations relationships and dependencies relationships as the cloned configuration.

It is worth mentioning that the 'Clone' operation is **not recursive**: related configurations are not recursively cloned, where only new relationships with existing ones are created.

The 'clone' operation can be activated using the 'clone' button in the toolbar or through the corresponding item in the contextual menu of the selected configuration.

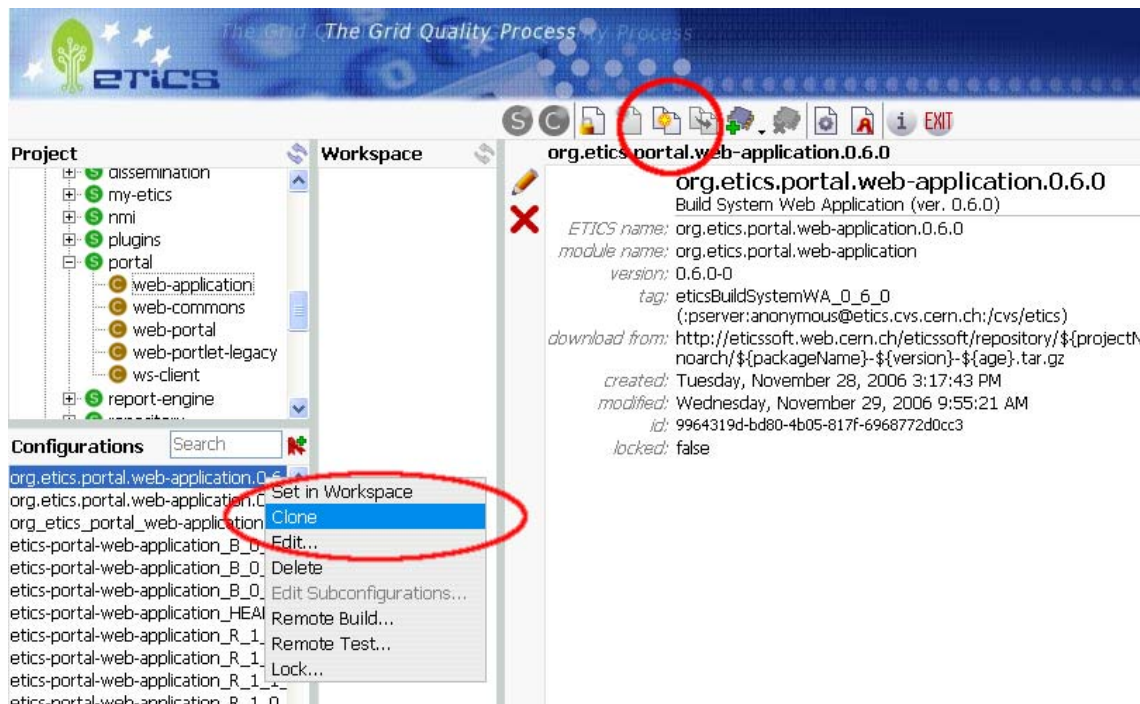


Figure 27 - Cloning Configurations

Locking Configurations

The 'Lock' operation makes a configuration no longer modifiable. In particular, its metadata, commands, properties, environment, dependencies and sub-configurations are 'frozen' to their current status.

For the lock to be effective, children configurations and dependencies must also be locked. For the children there are generally no authorization issues because the grant to lock is propagated to children items; for the dependencies, on the other hand, the user should have 'locking' privilege on the dependency configuration. If one of the related configurations cannot be locked, the whole lock operation will fail.

The 'lock' operation can be activated by using the 'lock' button in the toolbar or through the corresponding item in the contextual menu of the selected configuration.

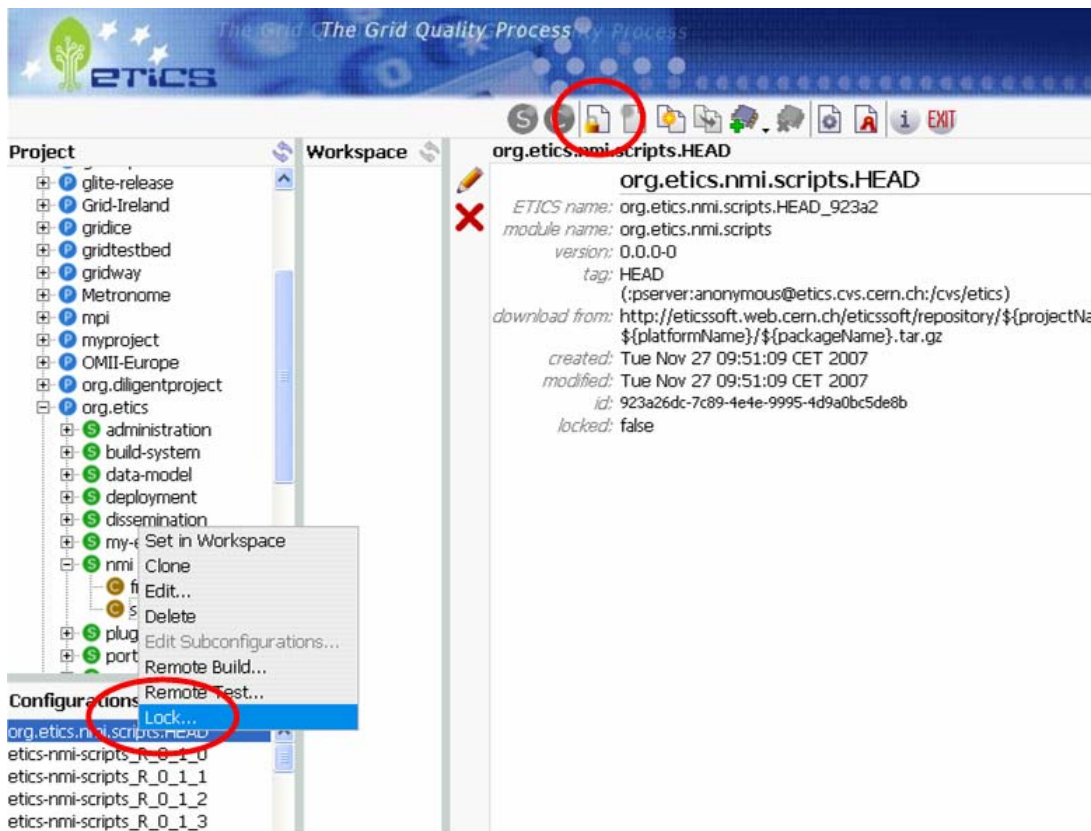


Figure 28: Locking configuration

Subsystem and Component configurations are generally built in the context of a Project/Subsystem configuration from which they inherit properties and from which dynamic dependencies are resolved. In order to reproduce the build in this context without locking the Project/Subsystem configuration itself, it's possible to specify a 'context' configuration to use when locking. Specifying a context allows building a locked configuration alone as if it was built in the context of an ancestor configuration, thus providing all inherited properties and resolved dependencies.

In the WA, the context configuration can be selected by using a drop down list.

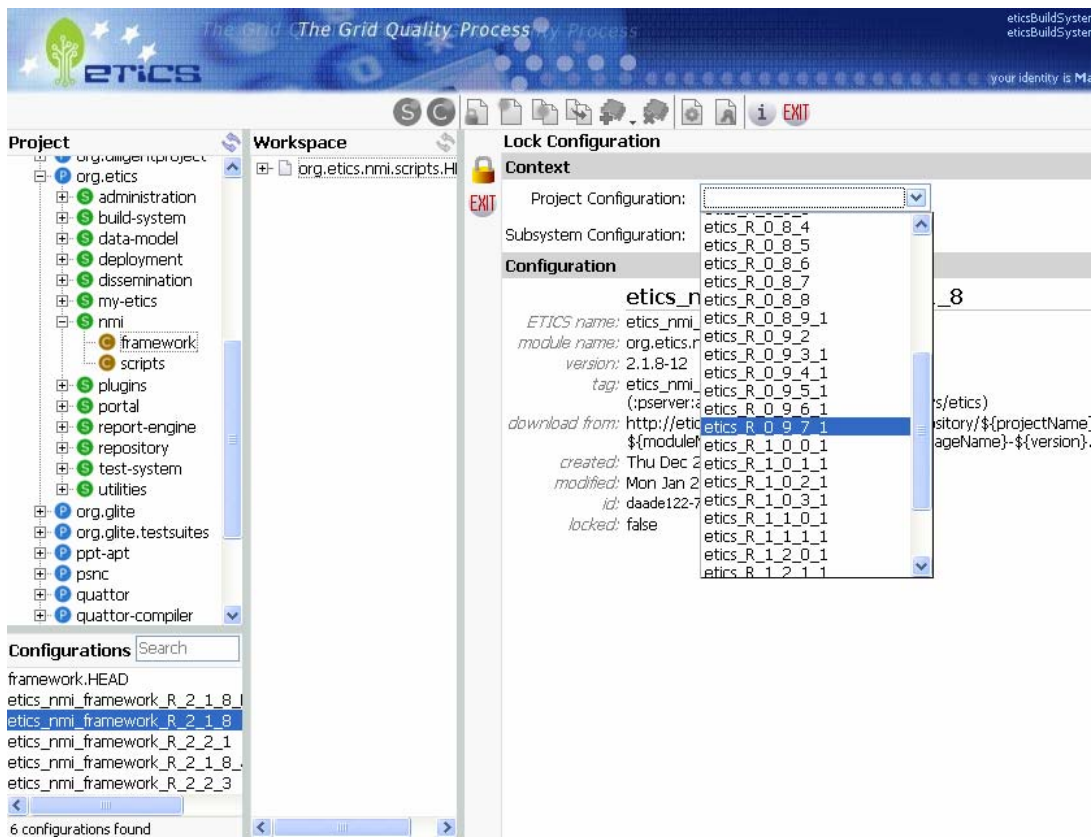


Figure 29: Selection of context configuration for locking

Note that if the configuration being locked is not a sub-configuration of project or subsystem configuration, the 'Context' section of the above shown form is not displayed.

7.2.3. Editing Sub-Configuration Relationships

The edit functionality for Sub-Configuration relationships is triggered using the 'edit sub-configuration' button in the toolbar or through the corresponding item of the popup-menu; this can be done either in the *workspace* or in the *configuration list*.

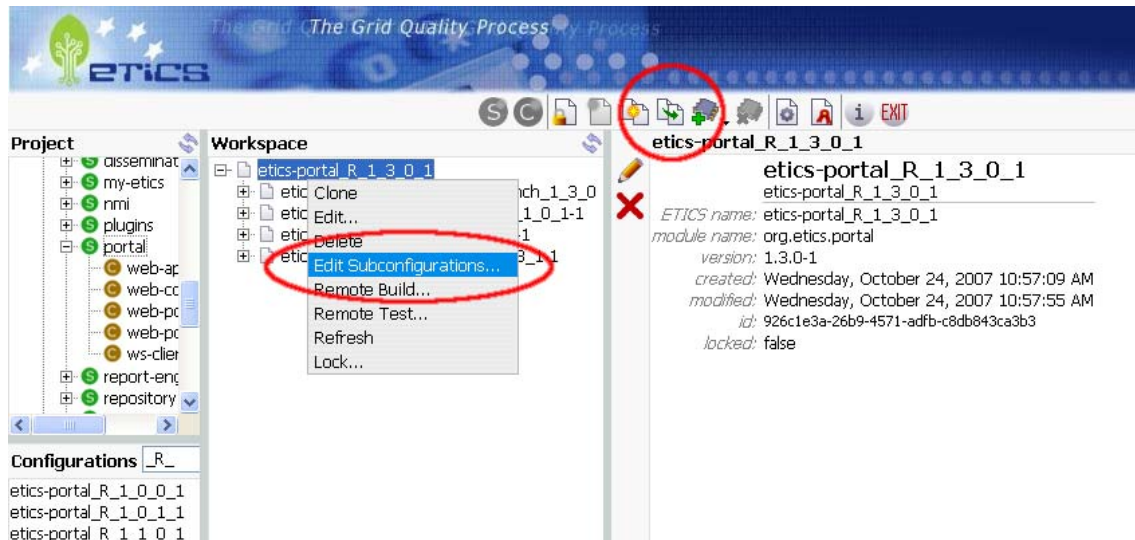


Figure 30: Editing sub-configuration relationships

The *edit area* is then updated with a view showing a list of module-configuration association. The information provided in the view is actually the same as the one provided by the configuration tree.

Beside each module name, a drop-down list allows to select a configuration from a list. After the selection, the chosen configuration will then appear in the module-configuration association, replacing any previously existing configuration. At the same time, the configuration tree on the left is also updated.

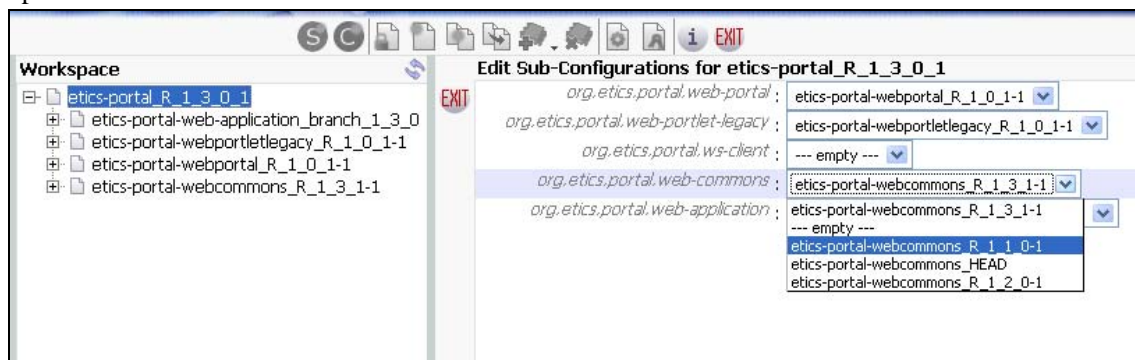


Figure 31 - Editing sub-configuration relationships

Removing Sub-Configurations

Sub-configuration relationships can also be removed by using the drop-down list in the sub-configuration editor: this is done by selecting the entry labelled '—empty—'. The previously-associated configuration will then be removed from configuration tree.

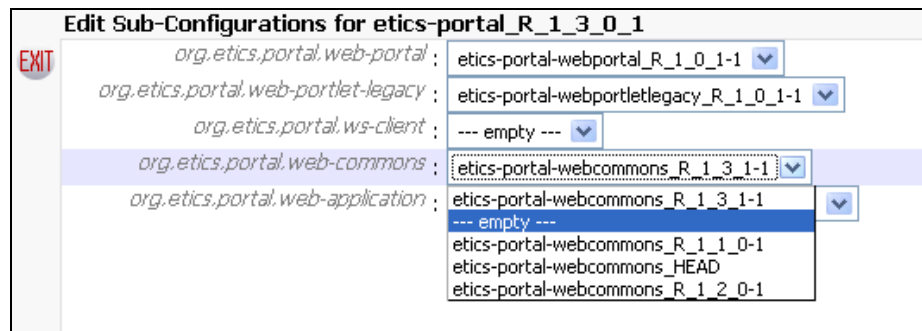


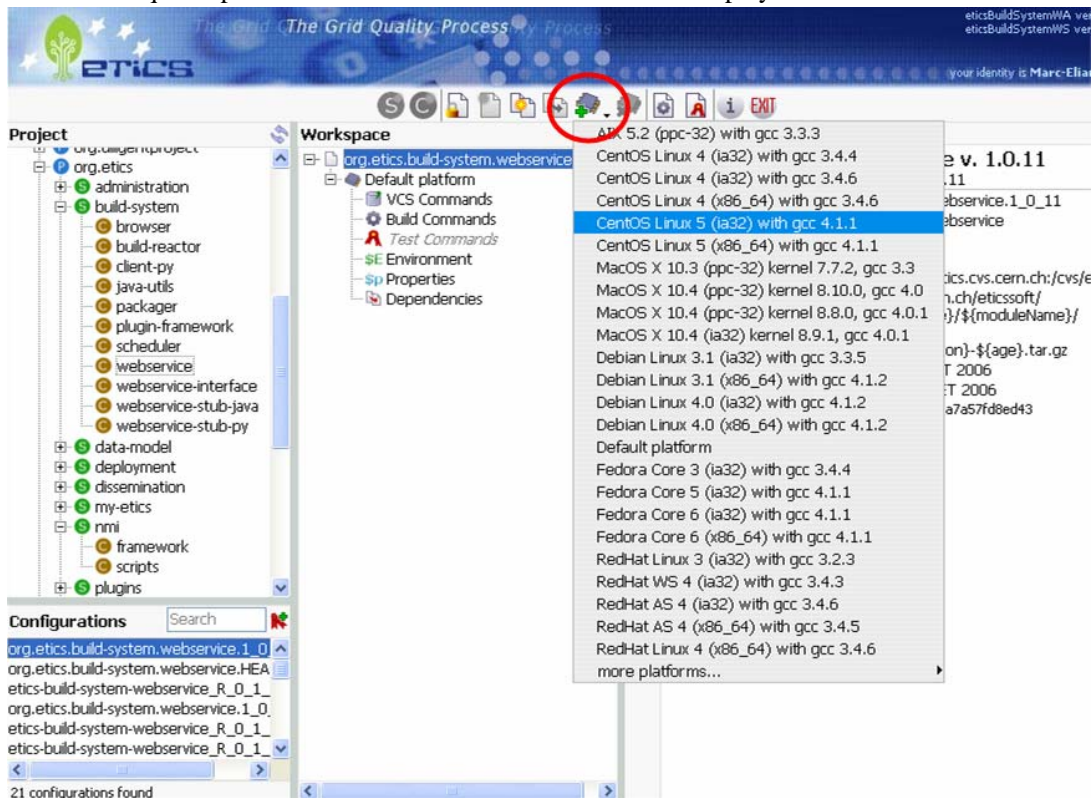
Figure 32: Removing sub-configuration menu

Note that *removing* a sub-configuration is different from *deleting* it, since in the case of delete the configuration is itself deleted, instead of simply removing the sub-configuration relationship.

7.2.4. Attaching Platforms to Configurations

With the only exception of Sub-Configurations, most configuration-related entities (e.g. commands, properties, dependencies, etc.) are platform-dependent. Thus, before creating any of such entities, a platform has to be selected.

Adding the support for a specific platform, among those available in the ETICS system, is done selecting a configuration in the workspace and then clicking the 'Attach Platform' button in the toolbar. The required platform can then be selected from the list displayed¹:



¹ Some platform entries might be greyed-out; this means that the support for the platform is already present for the selected configuration. This can be verified by expanding the configuration node in the tree.

Figure 33: Attaching platforms menu

After the selection, a new platform node is created under the configuration node. This node is the starting point for creating further platform-dependent entities.

7.2.5. Editing Commands

Commands (i.e. VCS Commands, Build Commands and Test Commands) can be modified using the 'edit' button in the main panel or through corresponding item in the contextual menu of command node. If a command is not defined for a platform, the corresponding node is rendered in italic style and gray colour. In this case it's sufficient to click on them to start editing.

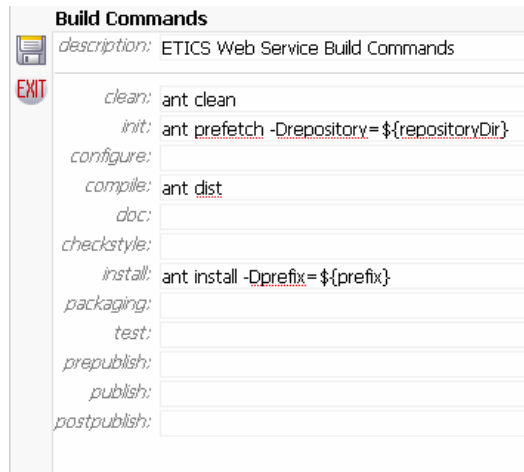


Figure 34: Editing Build Command form

When finished, click 'save' to store the command metadata or 'exit' to abort the action.

Commands can be removed clicking the 'remove' button in toolbar or by using the 'delete' entry from the context-menu associated to the command node.

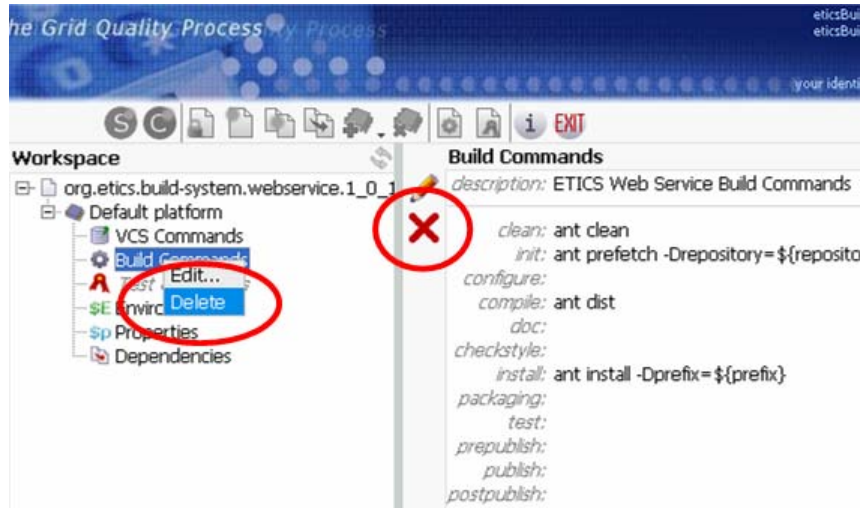


Figure 35: Deleting Build Commands

7.2.6. Editing Properties and Environment

For each configuration and platform, a set of properties and environment variables can be specified. When a new platform is attached to a configuration those sets are initially empty. They can be created selecting the properties or environment nodes, and then clicking the ‘edit’ button.

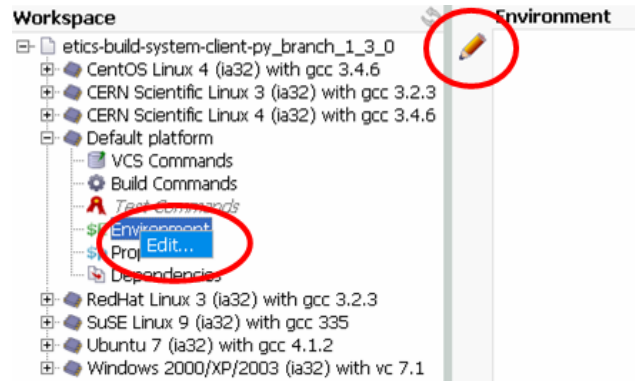


Figure 36: Editing environment variables menu

The details area on the right will then display a list of key-value pair.

New properties/environment can be created clicking on the ‘new property/environment’ button in the vertical toolbar. A form asking for property/environment name and value will appear in the editor.

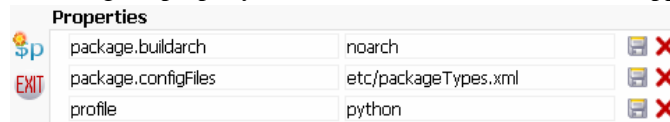


Figure 37: Editing property form

Existing properties/environment can also be modified by simply updating the value of fields.

A newly created entry as well as a modified one will display a little orange bar on their left side. That means that there are unsaved changes for the entry. To persist the changes click the ‘save’ button associated to the entry; alternatively, the ‘cancel’ button will undo either the modification.

7.2.7. Editing Dependencies

For each configuration and platform, a set of dependencies can be specified. When a new platform is attached to a configuration the dependency set is initially empty. New dependencies can be specified clicking the ‘edit’ button in the toolbar or selecting the ‘edit’ item from the context-menu associated to the dependencies node.

The details area on the right will then display a list of dependencies (if any).

When in edit mode, new configurations can be added as dependencies, by first selected from either the configuration list, the workspace¹ or the project tree. Details of the selected configuration will be shown in a floating box. To set this configuration as dependency, click the ‘add as dependency’ button in the vertical toolbar of the floating box. As a consequence the new configuration will appear in the dependency list.

¹ The content of the configuration tree and configuration list can be updated as usual

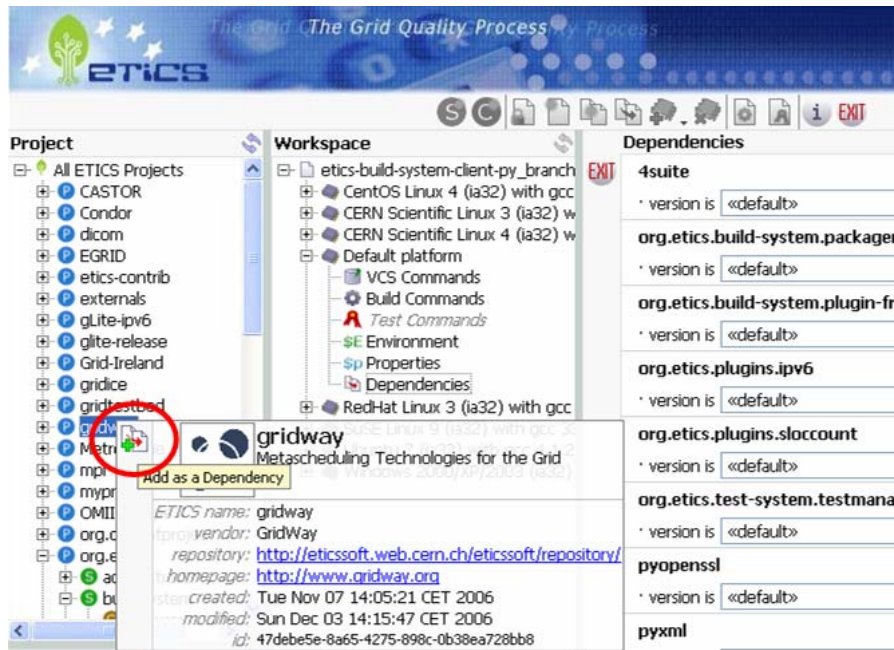


Figure 38: Add dependency floating box

Selecting a module from the project tree will result in adding a dynamic dependency. At this point, the dynamic dependency can be defined in terms of version, with or without an inequality, as shown in the figure below.

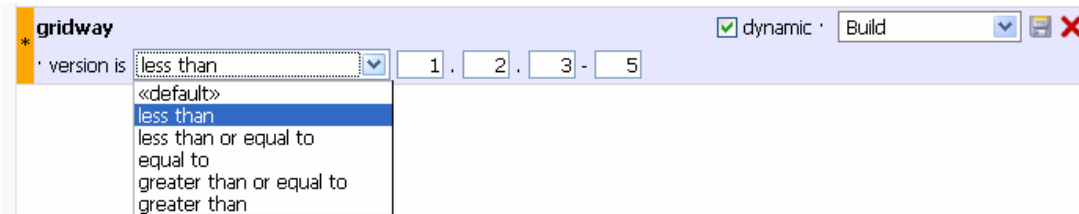


Figure 39: Dependency definition by version

Dependencies selected from the configuration list and the workspace are automatically added as static dependencies.

Existing entries can be modified; in this case, an orange bar is put on the left of the dependency telling that there are unmodified changes. To persist changes click the 'save' button.

Individual dependencies can also be removed by using the 'remove' button on the right side of each entry.

7.3. How to Edit with the Command-Line Client

This section describes how the edit commands work. These commands are applied to all the objects part of the ETICS data model described in Section 1.2 (i.e., modules, configurations, resources, platforms, users, and role). They support the following operations: *add* that registers a new object; *clone* creates a new copy of an existing configuration (only applicable for the configurations); *remove* that deletes an existing object; *modify* that updates object information and finally *prepare* that serialise a module of configuration in an *ini* formatted file.

The parameters that characterize an object in the operations add and modify, are provided in three different modes:

1. interactively, by requesting the user to provide the needed information;
2. by using a ini file;
3. by giving information in the command line.

Interactively

One of the modes to add and modify the objects of ETICS is to provide interactively the information to the system. The interactive mode is used by default when neither the ini file nor the `--param` option is given. The user is then requested to provide at the command-line values for the attributes of the object that have to be added/modified. Each attribute is shown in alphabetical order following this syntax:

```
<name-of-the-attribute>[<default-value>]:
```

The `name-of-the-attribute` is the name used by ETICS to describe a feature of the object (see the tables in Section 2 in order to have the list of the attributes for each type of object). The `default-value` is the value that the system will give to the attribute if the user doesn't change it. It could be the current value of the attribute if the operation is `modify` or a default value inherited by its parents if the operation is `add`. If the `default-value` is *None*, it means that no value is specified for that attribute.

After having printed a line like the previous one, the system waits for an input by the user. The possibilities of input by the user are:

- to press *Enter*, in this case the user doesn't want to change the value of the attributed and the one between square brackets is used
- to write the new value and then press *Enter*, in this case the new value is used.

If a name is followed by `***`, it means that a value is required and that the user must provide a value. Some attributes cannot be changed because they are related to other objects - e.g. the attribute `moduleName` for a configuration. In this case ETICS shows the name of the attribute followed by its value but it doesn't wait for an input and continues the execution to the next attribute.

Once all the attributes are processed, the system saves the new values. If the execution is stopped before the last attribute, all the changes are discarded.

This schema is used for all type of objects and also for the commands.

Via Ini File

Another way to provide parameters in the ETICS infrastructure is by using the ini mode during the add and modify operations. In order to use this mode the operation `prepare`, only applicable to module, configuration, resource and platform objects, has to be used to create an ini file in the user workspace. If the related object already exists in the ETICS data model, the file created will contain the current information; otherwise, the file created will contain default values. The default name of the ini file has the form `<object-type>-<object-name>.ini`, for example,


```
Component-org.glite.wms.common.ini
Configuration-org.glite.wms.common.HEAD.ini
Resource-joda.cnaf.infn.it
Subsystem-org.glite.wms.ini
```

Below you will find a partial schema of the ini file. As you can see, there are several sections: the first one is common in all the object ini files; the second section is only present in the module ini file; all the other sections are presented in the configuration ini file, and the last section is not included in the component configuration ini file. The dynamicDependency entry supports two syntaxes: the former specifies just the dependency type by the scope, the latter adds a constraint

```
[<Object-type>-<configuration-name>]
...
[Parent]
<parent-type>=<parent-name>
[Platform-<platform-name>:BuildCommand]
...
[Platform-<platform-name>:TestCommand]
...
[Platform-<platform name>:VcsCommand]
...
[Platform-<platform name>:Property]
;var1 = None
[Platform-<platform name>:Environment]
;var1 = None
[Platform-<platform name>:StaticDependency]
;<project-name>|<module-name> = <conf-name>,<dependency-type>
[Platform-<platform name>:DynamicDependency]
;<project-name>|<module-name> = <scope>
;<project-name>|<module-name> = <module-name><condition><version>,<scope>
[Hierarchy]
;<children-module-name> = <children-conf-name>
```

The operation `add` performs the insertion of new objects in the ETICS data model and it is applicable to module, configuration, platform and resource objects. For a module object, the operation automatically creates a module configuration called `<module-name>.HEAD`. The operation `clone` performs a copy of an existing configuration. It is only applicable to *configuration* objects. The operation `modify` updates an existing object and it is applicable to *module*, *configuration*, *platform*, and *resource* objects. The operation `rename` changes the name of an existing module or configuration. It is only applicable to *module* and *configuration* objects. The operation `remove` performs the deletion of an existing object and all its children. It can fail if the object itself (or its children) is used by other objects (e.g. the configuration is a dependency of another configuration or the configuration is specified as a child of the main configuration). Currently, these operations first commit the changes remotely and then locally. This behaviour is being changed in a future release,

where changes will only take place locally, and an explicit command will be required to commit the changes on the server.

7.4. THE ETICS-MODULE COMMAND

The ETICS command to edit a module is called `etics-module`. The syntax of the command is

```
etics-module <operation> [options] [<module-name>]
```

The following table shows the options:

Table 13: etics-module command options

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-I, --input <filename></code>	Specify the filename from which to get module parameters (ini file)
<code>-o, --output <filename></code>	Specify the filename where to copy module parameters (ini file)
<code>-f, --force</code>	If a destination file already exists, overwrite it without confirmation
<code>--param <param-value></code>	Allow user to specify parameters of the object module
<code>--parent <parent-name></code>	Specify the parentName of the specified module. It is mandatory during the operations add and remove of a component that belongs to a subsystem
<code>--component</code>	Specify that the moduleName is a component. It is mandatory during the operation add and remove
<code>-s, --subsystem</code>	Specify that the moduleName is a subsystem. It is mandatory during the operation add and remove
<code>--new-module <new module-name></code>	Specify a new name for an existing module
<code>--noask</code>	Do not ask the user to confirm the deletion. It is valid only for the remove operation
<code>--autocommit</code>	Commit module information first in the DB and then in the user work area. Its default value is True.
<code>--noautocommit</code>	Commit module information in the user work area. Its default value is False. To commit module information in the DB, the command <code>etics-commit</code> has to be used.
<code>--new</code>	Create a ini file for a new module.
<code>--verbose</code>	Print verbose messages.

During the operation `add` and `modify`, the *module-name* is mandatory in the following two cases: 1. the option `-i, --input` is not used; 2. the option `-i, --input` is used and the name of the module has to be changed.

7.4.1. How to prepare a module

A module template file in the ini format is created using different syntaxes of the command `et:ics-module` according to the type of module in the ETICS metadata store. This operation can be applied to module which type is a Project, or a Component or a Subsystem.

In order to create a component ini file which parent is the project, the ETICS command to be run is:

```
et:ics-module prepare --component <module-name>
```

This command serialises a module into a file called *Component-<module-name>.ini* which body contains either default values if the module doesn't exist, or the current values of the module if it already exists in the ETICS metadata store. For instance, if a user *prepares* an ini file for a new component called *voms.api*, which parent is the project called *OMII-Europe*, the file *Component-voms.api.ini* will be created, and its body will be:

```
[Component-voms.api]
licenseType=None
displayName=Voms.api
description=
repository=http://et:icssoft.web.cern.ch/et:icssoft/repository/
download=None
vendor=The OMII-Europe Consortium
packageName=None
homepage=None
vcsroot=None

[Parent]
Project=OMII-Europe
```

In order to create a component ini file which parent is a subsystem, the ETICS command to run is:

```
et:ics-module prepare --parent <subsystem-name> --component <module-name>
```

This command serialises a module into a file called *Component-<module-name>.ini* which body contains either default values if the module doesn't exist, or the current values of the module if it already exists in the ETICS metadata store. For instance, if a user *prepares* a ini file for an existing component called *voms.api-cpp*, which parent is the subsystem called *voms*, the file *Component-voms.api-cpp.ini* will be created, and its body will be:

```
[Component-voms.api-cpp]
licenseType=EGEE license
displayName=voms.api-cpp
description=CPP API for VOMS
repository=:pserver:nonymous@glite.cvs.cern.ch:/cvs/glite
```



```
download=None
vendor=The OMII-Europe Consortium
packageName=voms.api-cpp
homepage=http://voms.forge.cnaf.infn.it
vcsroot=:pserver:nonymous@glite.cvs.cern.ch:/cvs/glite
```

```
[Parent]
Subsystem=voms
```

In order to create a subsystem ini file which parent is the project, the ETICS command to be run is:

```
etics-module prepare -s <module-name>
```

This command serialises a module into a file called *Subsystem-<module-name>.ini* which body contains either default values if the module doesn't exist, or the current values of the module if it already exists in the ETICS metadata store. For instance, if a user *prepares* a ini file for an existing subsystem called *voms*, which parent is the project called *OMII-Europe*, the file *Subsystem-voms.ini* will be created, and its body will be:

```
[Subsystem-voms]
vendor=The OMII-Europe Consortium
displayName=voms
description=voms
repository=:pserver:nonymous@glite.cvs.cern.ch:/cvs/glite
vcsroot=:pserver:nonymous@glite.cvs.cern.ch:/cvs/glite
```

```
[Parent]
Project=OMII-Europe
```

7.4.2. How to add a module

The command to add a module in the ETICS infrastructure has different syntaxes according to the way the parameters are provided. In addition this operation automatically creates a configuration, called *<module-name>.HEAD* for the new module. This operation can be applied to a module which type is a Component or a Subsystem.

Interactively

The command to add a module providing its parameters interactively is one of the following:

- `etics-module add --component <module-name>`
in order to add the component *<module-name>* to the project;
- `etics-module add --subsystem <module-name>`
in order to add the subsystem *<module-name>* to the project;

- `etics-module add --parent <parent-name> --component <module-name>`

in order to add the component `<module-name>` to the parent `<parent-name>`.

Via ini file

The command to add a module providing its parameters via ini file is one of the following:

```
etics-module add -i Component-<module-name>.ini --component <module-name>
```

or

```
etics-module add -i Component-<module-name>.ini
```

in order to add the component `<module-name>` to the project;

```
etics-module add -i Subsystem-<module-name>.ini -s <module-name>
```

or

```
etics-module add -i Subsystem-<module-name>.ini
```

in order to add the subsystem `<module-name>` to the project;

- `etics-module add -i Component-<module-name>.ini --parent <parent-name> --component <module-name>`

or

```
etics-module add -i Component-<module-name>.ini
```

in order to add the component `<module-name>` to the parent `<parent-name>`.

Via command line

The command to add a module providing its parameters via command line is one of the following:

```
etics-module add --param description="new module" --component <module-name>
```

in order to add the component `<module-name>` to the project;

```
etics-module add --param description="new module" --subsystem <module-name>
```

in order to add the subsystem `<module-name>` to the project;

```
etics-module add --param description="new module" --parent <parent-name> --component <module-name>
```

in order to add the component `<module-name>` to the parent `<parent-name>`.

7.4.3. How to modify a module

The command to modify a module in ETICS has different syntaxes according to the way the parameters are provided. This operation can be applied to all module types – i.e. *project*, *subsystem* or *component*.

Interactively

The command to modify a module providing its parameters interactively is one of the following:

```
etics-module modify <module-name>
etics-module modify --subsystem <module-name>
etics-module modify --component <module-name>
```

Via ini file

The command to modify a module providing its parameters via ini file is one of the following:

```
etics-module modify -i Component-<module-name>.ini
etics-module modify -i Subsystem-<module-name>.ini
etics-module modify -i Project-<module-name>.ini
```

Via command line

The command to modify a module providing its parameters via command line is one of the following:

```
etics- module modify --param description="new module" <module-name>
etics-module modify --param description="new module" --subsystem
<module-name>
etics-module modify --param description="new module" --component
<module-name>
```

7.4.4. How to rename a module

The command to rename a module in ETICS has different syntaxes according to the way the parameters are provided. This operation can be applied to all module types – i.e. *project*, *subsystem* or *component*.

Via command line

The command to modify a module providing its parameters via command line is one of the following:

```
etics- module rename --new-module <new module-name> <module-name>
etics-module rename --new-module <new module-name> --subsystem
<module-name>
etics-module rename --new-module <new module-name> --component
<module-name>
```

7.4.5. How to remove a module

The command to remove a component module is:

Via command line

```
etics-module remove --component <module-name>
```

The command to remove a subsystem module is:

Via command line

```
etics-module remove -s <module-name>
```

This operation can be applied to module which type is a Component or a Subsystem.

7.5. The etics-configuration Command

The ETICS command to edit a module is called `etics-configuration`. The syntax of the command is

```
etics-configuration <operation> [options] [-c configuration-name]
[configuration-name-cloned] <module-name>
```

The following table shows the options:

Table 14: etics-configuration command options

Option	Description
-h, --help	Show the usage instructions
-i, --input <filename>	Specify the filename from which to get configuration parameters
-o, --output <filename>	Specify the filename where to copy configuration parameters
-f, --force	If a destination file already exists, overwrite it without confirmation
--param <param-value>	Override existing or define new properties
--new-config <new configuration-name>	Specify a new name for an existing configuration. It defines the name of either a cloned configuration or a renamed configuration.
--new-input <filename>	Specify the filename from which to get new configuration information.
--forceremoval	Force the removal when the configuration is linked by other configuration. Other configuration could be destroyed by using this option. It is valid only for the remove operation.
--noask	Do not ask the user to confirm the deletion. It is valid only for the remove operation
-p, --project <project-name>	Specify the project name to which the configuration belongs.
--parent-config <parent configuration-name>	Specify the configuration name of the parent which the configuration to be locked belongs to.
--recursive	Force the recursion for clone operation if the configuration belongs to a project or a subsystem. Its default value is False.
--recursive-suffix <suffix-value>	Specify a suffix to append during a clone operation. Its default value is 'cloned'.
--autocommit	Commit configuration information first in the DB and then in the user work area. Its default value is True.
--noautocommit	Commit configuration information in the user work area. Its default value is False. To commit configuration information in the DB, the command <code>etics-commit</code> has to be used.
--new	Create a ini file for a new configuration.
--verbose	Print verbose messages.

During the operation add and modify, the `<module-name>` and the option `-c <configuration-name>` are mandatory in the following two cases: 1. the option `-i, --input` is not used; 2. the option `-i, --input` is used and the name of the configuration has to be changed. The option `-p, --project` can be used during the operations prepare, add and modify, if the configuration belongs to a module of a different project respect to what get by using the command `et:ics-get-report`. In this case, the changes done by using the operations add and modify will not be saved in the database. Therefore it is suggested to use it together with the option `--noautocommit` in order to avoid users to obtain an error message from the command `et:ics-configuration`.

7.5.1. How to prepare a configuration

The command to serialise a configuration into an ini formatted file is:

```
et:ics-configuration prepare -c <configuration-name> <module-name>
```

This command serialises a configuration into a file called *Configuration-<configuration-name>.ini* which body contains either default values if the configuration doesn't exist, or the current values of the configuration if it already exists in the ETICS metadata store

For instance, if a user *prepares* an ini file for a new configuration called *module.TEST*, the file *Configuration-module.TEST.ini* will be created, and its body will start with:

```
[Configuration-module.TEST]
```

For instance, if a user *prepares* an ini file for an existing configuration such as *voms.HEAD*, associated with the subsystem *voms*, the file *Configuration-voms.HEAD.ini* will be created, and its body will be:

```
[Configuration-voms.HEAD]
moduleName = voms
projectName = org.glite
displayName = voms.HEAD
description = None
age = 0
tag = HEAD
path=${projectName}/${moduleName}/${version}/${platformName}/${packageName}-${version}-${age}.tar.gz
version = 1.0.0
```

```
[Platform-None:BuildCommand]
:init = None
:install = None
:checkstyle = None
:clean = None
:displayName = None
```

```
;compile = None
;configure = None
;prepublish = None
;packaging = None
;test = None
;doc = None
;description = None
;publish = None
;postpublish = None

[Platform-None:TestCommand]
;clean = None
;init = None
;displayName = None
;description = None
;test = None

[Platform-None:VcsCommand]
;branch = None
;tag = None
;displayName = None
;description = None
;checkout = None
;commit = None

[Platform-None:Property]
;var1 = None

[Platform-None:Environment]
;var1 = None

[Platform-None:StaticDependency]
;<project-name>|<module-name> = <conf-name>,<scope>

[Platform-None:DynamicDependency]
;<project-name>|<module-name> = <scope>
;<project-name>|<module-name> = <module-
name><condition><version>,<scope>

[Hierarchy]
voms.all = voms.all.HEAD
voms.api-cpp = voms.api-cpp.HEAD
```

Reading this file, we observe that the configuration `voms.HEAD` related to the subsystem `voms` contains two sub-configurations, one of the component `voms.all` and the other one of the component `voms.api-cpp`. We also observe that this configuration does not contain any platform.

For instance, if a user *prepares* an ini file for existing configuration such as `voms.api-cpp.HEAD`, associated with the component `voms.api-cpp`, the file *Configuration-voms.api-cpp.HEAD.ini* will be created, and its body will be:

```
[Configuration-voms.api-cpp.HEAD]
version = 1.0.0
moduleName = voms.api-cpp
projectName = org.glite
displayName = voms.api-cpp.HEAD
description = None
age = 1
tag = Unused
path =
${projectName}/${moduleName}/${version}/${platformName}/${packageName}
e}-${version}-${age}.tar.gz

[Platform-default:BuildCommand]
postpublish = None
packaging = None
displayName = None
description = None
doc = None
prepublish = None
publish = None
compile = mkdir -p build; cp -R stage/lib build
init = mkdir -p stage/lib; cp ${stageDir}/lib/libvomsapi* stage/lib
install = cp -R build/lib ${prefix}
clean = None
test = None
checkstyle = None
configure = None

[Platform-default:VcsCommand]
tag = None
displayName = None
description = None
branch = None
commit = None
checkout = echo "VCS for voms.api-cpp"; mkdir -p voms.api-cpp
```

```
[Platform-default:TestCommand]
;clean = None
;init = None
;displayName = None
;description = None
;test = None

[Platform-default:Property]
;var1 = None

[Platform-default:Environment]
;var1 = None

[Platform-default:StaticDependency]
OMII-Europe|voms.all = voms.all.HEAD,BR

[Platform-default:DynamicDependency]
;<project-name>|<module-name> = <scope>
```

Reading this file, we observe that the configuration *voms.api-cpp.HEAD* related to the component *voms.api-cpp* is associated to the platform *default*. For that Platform, this configuration has one BuildCommand, one VcsCommand, and one StaticDependency with the configuration of the component *voms.all*.

It is possible to have more than one platform associated to the same configuration. In this case the ini file will contain all information presented in the ETICS server data-store.

7.5.2. How to add a configuration

The command to add a configuration in the ETICS infrastructure has different syntaxes according to the way the parameters are provided.

Interactively

The command to add a configuration providing its parameters interactively is:

```
etics-configuration add -c <configuration-name> <module-name>
```

After having added the configuration in the standard interactive mode, the system shows a menu with different options in order to allow the user to associate platforms to the new configuration. This menu will be explained in the section 6.5.4.

Via ini file

The command to add a configuration providing its parameters via ini file is:

```
etics-configuration add -i Configuration-<configuration-name>.ini
```


Via command line

The command to add a configuration providing its parameters via command line is:

```
etics-configuration add --param description="new configuration" -c  
<configuration-name> <module-name>
```

7.5.3. How to clone a configuration

The command to clone a configuration is:

Via command line

```
etics-configuration clone -c <configuration-current-name>  
<configuration-new-name> <module-name>
```

But it is also possible to do as follows when the configuration belongs to a subsystem or a project:

```
etics-configuration clone --recursive -c <configuration-current-  
name> <configuration-new-name> <module-name>
```

The configuration itself with its children will be cloned and their new name will contain the suffix 'cloned'.

```
etics-configuration clone --recursive --recursive-suffix <suffix-  
value> -new-config <new configuration-name> -c <configuration-  
current-name> <configuration-new-name> <module-name>
```

The configuration itself with its children will be cloned and their new name will contain the suffix <suffix-value>.

7.5.4. How to modify a configuration

The command to modify a configuration in the ETICS infrastructure has different syntaxes according to the way the parameters are provided.

Interactively

The command to modify a configuration providing its parameters interactively is:

```
etics-configuration modify -c <configuration-name> <module-name>
```

The operation `modify` of a configuration also allows the user to edit the associated platforms, the commands, dependencies, environments and properties linked to it and to set the hierarchy of the configuration (i.e. specify children configurations).

This command, when used interactively, shows a menu with different options:

1. **Modify the general configuration parameters:** this option is used for changing the attributes of the configuration.
2. **Support a new platform for this configuration:** this option is used for supporting a new platform. The system displays the platforms currently supported for the configuration and asks

for the name of the new one. If an already supported platform or a non existing one is given, an error is raised and it goes back to the main menu.

If the name of the platform is correct, the user is requested to insert the value for the types of objects to add to the platform (i.e. commands, dependencies, properties and environment variables). None of those are mandatory but at least one of them must be added. The build, vcs and test commands are unique for each configuration and platform and therefore only one object per type can be inserted. For the other objects the system accepts multiple insertions; it move to the following type if a blank or inconsistent value is given.

3. **Modify configuration options (build/vcs/test commands, properties, environments, dependencies):** this option is used for modifying the values of the objects linked to a platform already associated to the configuration. It lists the associated platforms and the user must choose one of them. Then it shows the names of the objects defined for that platforms and allows the user to add/modify/remove them.
4. **Modify the configuration's children in the configuration's hierarchy:** this option is used for defining the hierarchy of the current configuration. The children, if any, are shown and the user can delete them. It is also possible to add new children choosing them among the configurations of the modules that are children of the module of the configuration. For example, if the user is modifying a configuration of a subsystem that has two components as children, this option allows to define a child configuration for each of the components.
5. **Exit**

Via ini file

The command to modify a configuration providing its parameters via ini file is:

```
etics-configuration modify -i Configuration-<configuration-name>.ini
```

By default the information contained in the ini file is considered to be complete, this means that if there is more information in the server-side metadata store than in the ini file, once this command is executed only the information contained in the ini file will remain in the metadata store.

Via command line

The command to modify a configuration providing its parameters via command line is:

```
etics-configuration modify --param description="new configuration" -  
c <configuration-name> <module-name>
```

7.5.5. How to remove a configuration

The command to remove a configuration is:

```
etics-configuration remove -c <configuration-name> <module-name>
```

7.5.6. How to rename a configuration

The command to rename a configuration is:

Via command line

```
etics-configuration rename --new-config <new configuration-name> -c <configuration-name>
<module-name>
```

But it is also possible to do as follows when the configuration belongs to a subsystem or a project:

```
etics-configuration rename --new-input Configuration-<new configuration-name>.ini -i Configuration-
<configuration-name>.ini
```

The configuration itself with its children will be renamed according what it is specified in the ini file Configuration-<new configuration-name>.ini.

7.5.7. How to lock a configuration

The command to lock a configuration is:

Via command line

```
etics-configuration lock --parent-config <parent configuration-name> -c <configuration-name>
<module-name>
```

7.6. THE ETICS-COMMIT COMMAND

The ETICS command to commit changes performed by using the commands `etics-module` and `etics-configuration` is called `etics-commit`. The syntax of the command is

```
etics-commit
```

8. TAGGING CONFIGURATIONS

8.1. Overview

Previous chapters described how to create configurations using the standard command-line client. During development and maintenance users may need a quick and simple way to add new configurations based on existing ones as a consequence of fixing code bugs or making minor modifications that do not require big changes in the configuration commands or properties. In this case the `etics-tag` command can be used.

8.2. The `etics-tag` Command

The `etics-tag` command creates a configuration from an existing one and optionally tags the code in the underlying Version Control System. This command is logically equivalent to the CVS `tag` command where this latter creates a new tag on an existing branch. The syntax of the command is:

```
etics-tag [options] [<module-name>]
```

The command has several options that allow controlling how the tagging is executed:

Table 15: `etics-tag` command options

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-c, --config <configuration-name></code>	The configuration name to be created from the current one. Required to create new configurations. If this option is used, the operation only applies to the specified module (implies <code>--nodeps</code>). If the <code><module-name></code> argument is provided, the configuration applies to the module, otherwise it applies to the current project.
<code>-t, --tag <tag></code>	The VCS tag string to be used for tagging. If this option is not specified the configuration name is used as tag string
<code>--config-version <version></code>	The version of the configuration to be created/updated in the form <code>x.y.z-r</code> .
<code>-i, --inputfile <file></code>	An input file in the ETICS INI format to be used. If this file is provided, the <code>--tag</code> and <code>--config-version</code> options are ignored (the values are taken from the file). The operation is performed on the current configuration of the specified module if the configuration name in the INI file matches it or a new configuration is created if the name doesn't match
<code>--prepare-childlist <file></code>	Using this option it is possible to create a template childlist file containing an entry for each component in the subsystem (and the subsystem itself) with the current configuration name, version and tag. This file can be edited and passed to the <code>--childlist</code> option to tag new versions of a subsystem
<code>--childlist <file></code>	A file containing a list of children to be tagged with the parent. Each line in the file has the format: module-name config-name x.y.z-age

<code>-u, --update</code>	If the configuration already exists, update it with the new values. If this option is not specified creating an existing configuration produces an error
<code>--notag</code>	Do not perform the actual VCS tag and configuration creation/update.
<code>--novcstag</code>	Do not perform the actual VCS tag, but perform the configuration creation/update.
<code>--nodeps</code>	Only tag the currently specified module (do not tag children)
<code>--continueonerror</code>	Continue when tag errors are encountered.
<code>--verbose</code>	Print verbose messages
<code>--version</code>	Display the current client version number.

8.2.1. How to Tag a Configuration

The standard command to tag a configuration is:

```
etics-tag -c <configuration-name> -t <tag> --config-version <x.y.z.-r> <module-name>
```

This command creates a new configuration of the specified module by cloning the current configuration in the local workspace (or HEAD if no configuration is present) using the new name, VCS tag and version. If the configuration is stored in a Version Control System and a *tag* target is defined in the *VCS Commands*, the code is also tagged in the VCS. If the module name is not specified, the command applies to the project.

If a more complex configuration has to be created as result of the tag operation, an INI file in the ETICS format can be supplied with the `-i` option. In this case the `--tag` and `--config-version` options are ignored if specified.

8.2.2. How to Tag a Configuration Tree

The standard command to tag a configuration tree is:

```
etics-tag -c <configuration-name> -t <tag> --config-version <x.y.z.-r> --childlist <filename> <module-name>
```

where `<module-name>` must be a subsystem name.

This command creates a new configuration of the specified subsystem by cloning the current configuration in the local workspace (or HEAD if no configuration is present) using the new name, VCS tag and version. If the configuration is stored in a Version Control System and a *tag* target is defined in the *VCS Commands*, the code is also tagged in the VCS. Additionally, the command parses the supplied child list file, tags if necessary each module listed in the file and attaches the configuration to the parent subsystem configuration. Note that the child configurations listed in the child list file don't have to be all new configurations to be tagged. If a configuration already exist, it is skipped or updated depending on the presence of the `--update` option. However, all listed



configurations are attached to the parent subsystem configuration no matter if they have been just tagged or not.

If the module name is not specified, the command applies to the project.

A helper option called `--prepare-childlist` can be used to create a template childlist file that can then be edited with new values for one or more entries. The command to generate this file is:

```
etics-tag --prepare-childlist <filename> -c <configuration-name>  
<module-name>
```

9. BUILDING CONFIGURATIONS

9.1. Overview

Previous chapters described how to get a project into a workspace and how to browse its structure and configurations and edit them by creating, modifying or deleting modules, configurations and related commands, properties and dependencies.

Once the information is registered in the ETICS system, we can use the ETICS client to execute builds and generate software packages and reports. This chapter will describe the commands used to perform these operations and how to control how builds are performed.

In the following section, it is assumed that a project has been inserted in the workspace and that a suitable configuration has been checked out.

9.2. The etics-build Command

The ETICS command to perform builds is called `etics-build`. The syntax of the command is:

```
etics-build [options] [<module-name>]
```

The command has several options that allow controlling how builds are executed. The following table shows the most common options, while more advanced options will be described in the following sections:

Table 16: etics-build command options

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-c, --config <configuration-name></code>	Define a specific configuration to be used instead of the default one, where default is the first configuration found in the store. This is normally not required unless multiple or non-default configurations of the selected module have been checked out
<code>-p, --property <property=value></code>	This option allows passing properties to the build process. If the property is already defined, its value is overridden by the value specified in the command-line. To pass multiple properties, use multiple <code>-p</code> options
<code>-e, --env <env=value></code>	This option allows passing environment variables to the build process. If the variable is already defined, its value is overridden by the value specified in the command-line. To pass multiple variables, use multiple <code>-e</code> options
<code>-t, --target <target-name></code>	Execute the build stopping at the specified target, If not specified the publish target is executed. Allowed targets are: clean, init, checkstyle, compile, test, doc, packager, publish, install
<code>--platform <platform-name></code>	Overwrite the local platform (useful for testing or when the local platform is not a valid ETICS platform, but it's compatible with one).
<code>--nobuild</code>	Do not perform the build, just print the sequence
<code>--continueonerror</code>	Do not stop building if an error is found
<code>--nodeps</code>	Only build the currently specified module (do not build children

	and dependencies)
--force	Force the build of unmodified modules
--cache	Enable the property cache. This option allows to build using information cached from a previous <code>etics-checkout</code> command. It's the default behaviour (no need to specify this option explicitly). This method is faster, but may fail if not all information needed for building is available in the cache
--nocache	Disable the property cache. If building with the cache fails, this option can be used to regenerate the properties and build sequence. It is safer, but slower

9.2.1. How to Build a Configuration

The standard command to build a configuration is:

```
etics-build <module-name>
```

This command builds the default configuration of the specified module. The module must be a module within the module structure checked out with the `etics-checkout` command. The default configuration is the first configuration found in the local metadata store.

When run by itself, the command builds the current configuration of the current project. Therefore

```
etics-build ➔ etics-build <current-project>
```

This assumes that a valid project configuration has been checked out or the command will fail prompting to check out the project.

If more than one configuration of the same module exists in the store, for example because a non-default configuration has been checked out (see Chapter 6 for more information about checking out configurations), the `-c` option can be used to select a specific configuration:

```
etics-build -c <configuration-name> <module-name>
```

9.2.2. Build Targets

The build targets are defined by the commands available in the Build Commands set associated to a configuration for the current or the default platform. The available targets are described in Section 1.2.4 - Commands and Properties.

The `etics-build` command executes the target in the following order:

init ➔ checkstyle ➔ compile ➔ test ➔ doc ➔ packaging ➔ publish

stopping after the execution of the target specified with the `-t` option or after *publish* if `-t` is not used. The *clean*, *install* and *configure* targets must be executed separately using the `-t` option.

If a target is not specified (i.e. left empty) it is normally simply skipped. However, there are a few special rules:

1. If *packaging* is not defined, the internal ETICS Packager is used. This is the recommended behaviour, unless you really have some special packaging tasks already defined for your code
2. If the *install* target is not defined, no packaging can be done with the ETICS Packager, since ETICS relies on your Build Command's *install* target to detect which files must be packaged.
3. If *publish* is not defined, the internal ETICS Publisher is used. It is however possible to mix the ETICS Publisher with custom publishing actions by using the command `${eticsPublisher}` anywhere in your *publish* target. For example:

```
publish = my_pre_publish_steps_here.sh;  ${eticsPublisher};  
my_post_publish_steps_here.sh
```

More information about the *packaging* and *publish* targets are given in following sections.

9.2.3. Forcing Build Execution

The standard behaviour of the `etics-build` command is to execute any given action once and then cache information on disk that prevents executing again the same action if nothing in the code has changed. In addition, the caching mechanism is aware of the target chaining, so if the *publish* target is executed first, then any attempt to run explicitly the other targets will report that there is nothing to do, since they have been already executed. This allows speeding up the operation of restarting a failed build, since all successfully executed steps will be skipped.

However in certain occasions it may be necessary to rerun the same action again, for example because something has changed in a place outside ETICS control or because the same action has to be repeated with different options. In this case, the build can be force using the `--force` option:

```
etics-build --force -c <configuration-name> <module-name>
```

9.3. The ETICS Packaging System

The ETICS Client comes with a built-in packaging system able to build distribution packages on several supported platforms in different formats¹ (e.g. tarballs, RPMS, debs, MSIs). In order to use the ETICS Packager the *install* target has to be defined, since it is used by the packager to identify the content of the package.

In order to activate the ETICS Packager simply leave the *packaging* target undefined. To disable the Packager and if no custom packaging action is needed, the packaging target must be set to an 'empty' command, like the shell *echo* command for example.

By default the ETICS Packager tries to create binary tarballs and RPMS, debs or MSIs (depending on the working platform¹) by using information collected during the build, such as run-time dependencies. The generated packages have the following naming conventions:

Binary tarball: `<module-name>-x.y.x-r.tar.gz`

¹ In the current version the supported packaging format are tarballs, RPMS and debs.

Source tarball: `<module-name>-x.y.x-r.src.tar.gz`
 Binary RPMS: `<module-name>-x.y.x-r.<distribution>.<architecture>.rpm`
 Source RPMS: `<module-name>-x.y.x-r.src.rpm`
 Binary debs: `<module-name>_x.y.x-r.<distribution>_<architecture>.deb`
 Source dsc: `<module-name>_x.y.x-r.dsc`

where distribution is a standard OS distribution name like slc3, rhel4, fc4, ubuntu7, etc and architecture is a cpu architecture name like i386, i686, x86_64, noarch/all, etc.

A number of options and built-in properties are available to customise the Packager behaviour. The possible options are listed in the following table:

Option	Description
<code>--createsource</code>	Create source tarball and source RPMS in addition to binary ones (this slows down the build due to extra compilation steps)
<code>--createdebbug</code>	Create debug RPMS in addition to binary ones (this slows down the build due to the extra compilation steps) ¹
<code>--nodistname</code>	Do not append the distribution name to the RPM revision (it has no effect if <code>--userspec</code> is used)
<code>--usetimestamp</code>	Append a timestamp to the RPM revision number (it has no effect if <code>--userspec</code> is used)
<code>--versioneddeps</code>	Use version information when setting package dependencies in RPMS (ex: <code>etics-client >= 1.0.0</code>). Cannot be used together with <code>--strictversioneddeps</code>
<code>--strictversioneddeps</code>	Use strict version information when setting package dependencies in RPMS (ex: <code>etics-client = 1.0.0</code>). Cannot be used together with <code>--versioneddeps</code>
<code>--userspec <path></code>	Use a user-defined spec file for RPM generation. If this option is used all other packaging options are ignored. <code><path></code> is relative to the module source location
<code>--usercontrol <path></code>	Use a user-defined control file for deb generation. If this option is used other packaging options are ignored. <code><path></code> is relative to the module source location
<code>--userchangelog <path></code>	Use a user-defined changelog file for the deb. If this option is not used, a default changelog file is created containing a single version info entry. <code><path></code> is relative to the module source location
<code>--userrules <path></code>	Use a user-defined rules for deb generation. <code><path></code> is relative to the module source location

Table 17: etics-build command packaging options

¹ Not supported in the current release

In addition to the command-line options, the following built-in properties can be specified either as properties attached to the configuration or as command-line properties (a complete list of all built-in ETICS properties can be found for reference in Appendix B at the end of the User Guide):

Property name	Default value if not specified	Type	Description
package.prefix	/opt/module-name	String	The default installation prefix for the generated RPMS
package.buildarch	Local platform/distribution specifier (slc3, rhel4, noarch, etc)	String	It can be used to override the default specifier. It is necessary to set it to 'noarch' to generate 'noarch' RPMS and 'all' debs
package.priority	optional	String	The value of the Priority field in the deb control file
package.section	devel	String	The value of the Section field in the deb control file
package.group	Unknown	String	The value of the Group tag in the RPM spec file
package.packager	ETICS	String	The value of the Packager tag in the RPM spec file and the Maintainer field in the deb control file
package.provides	Autodetected	String	The list of provides entries for the RPM spec file (overrides the autodetected list)
package.requires	Autodetected	String	The list of Requires entries for the RPM spec file (overrides the autodetected list)
package.depends	Autodetected	String	The list of Depends entries for the deb control file (overrides the autodetected list)
package.predepends		String	The list of Pre-Depends entries for the deb control file
package.obsoletes		String	The list of Obsoletes

package.replaces		String	entries for the RPM spec file The list of Replaces entries for the deb control file
package.conflicts		String	The list of Conflicts entries for the RPM spec file and deb control file
package.recommends		String	The list of Recommends entries for the deb control file
package.suggests		String	The list of Suggests entries for the deb control file
package.enhances		String	The list of Enhances entries for the deb control file
package.autoreqprov	Yes	String	Can be 'yes' or 'no'. If it is set to yes, rpm will try to automatically detect requires and provides from the packaged files and libs, otherwise will only use specified entries
package.userspec		String	Use a user-defined spec file for RPM generation. If this option is used all other packaging options are ignored. The value is a file path relative to the module source location
package.usercontrol		String	Use a user-defined control file for deb generation. If this option is used other packaging options are ignored. The value is a file path relative to the module source location
package.userchangelog		String	Use a user-defined changelog file for the

package.userrules		String	deb. If this option is not used, a default changelog file is created containing a single version info entry. The value is a file path relative to the module source location
package.configFiles		String	Use a user-defined rules for deb generation. The value is a file path relative to the module source location
package.configFilesNoreplace		String	A comma-separated list of files to be flagged as configuration files in the RPM spec file and in the deb conffiles file
package.suidFiles = <file-path[,file-path]>		String	A comma-separated list of files to be flagged as configuration files in the RPM spec file and in the deb conffiles file. In the case of the RPM spec file, the entries are flagged as %config(noreplace)
package.fileRights = <file-path@xxx[,file-path@xxx]>		String	Comma-separated list of files to be installed with permission mask 4555
package.tgz.name		String	Comma-separated list of files to be installed with permission mask xxxx (the mask can be different for each file)
		String	If the tarball name in the ETICS repository doesn't follow the standard ETICS conventions, this property can be used to specify a custom name. It only applies

			when downloading packages, not when creating packages
package.deb.name		String	If the deb name in the ETICS repository doesn't follow the standard ETICS conventions, this property can be used to specify a custom name. It only applies when downloading packages, not when creating packages
package.rpm.name		String	If the RPM name in the ETICS repository doesn't follow the standard ETICS conventions, this property can be used to specify a custom name. It only applies when downloading packages, not when creating packages
package.msi.name		String	If the msi name in the ETICS repository doesn't follow the standard ETICS conventions, this property can be used to specify a custom name. It only applies when downloading packages, not when creating packages

Table 18: packaging properties

9.3.1. How to Pass Installation Instructions to the Packager

As previously seen, it is necessary to pass installation instructions to the Packager using the *install* target in order to create packages. However, the installation command by itself is not enough, since typical installation procedures tend to install the software in locations that are not writable for non-privileged users. The ETICS Packager needs to have a way of redirecting the installation to a location of its choice. This is typically done by most installation procedures by specifying explicitly the installation prefix.

Since there are various conventions about how to pass this parameter depending on which installation method is used (autotools, ant, python distutils, etc), ETICS defines the following generic conventions: whatever method is used to pass the prefix, the value of the prefix must be set in the

commands as `${prefix}`. The ETICS Packager will then resolve this value to an internal location of its own. For example, using standard Autotools practices, this becomes:

```
init = ./configure
compile = make
install = make install --prefix=${prefix}
```

or using python distutils

```
compile = python setup.py build
install = python setup.py --prefix=${prefix}
```

and so on. Of course the same convention applies to completely custom installation scripts as long as it is possible to pass the installation root in some way, for example:

```
install = ./my_installation_script --install_location=${prefix}
```

9.4. The ETICS Publisher

The ETICS Client comes with pre-defined publishing conventions that allow harvesting of build and test artefacts (e.g. packages, logs, metrics, test reports) from the various modules into a common location. The ETICS Publisher uses the following conventions:

1. All checkout and build logs are stored in the reports directory of the workspace. The main log file is called:

```
build-status.xml
```

and contains summary checkout, build and test information for each module in the current build or test run. Detailed logs of the checkout, build and test operations of each modules are saved in files of the format:

```
[CHECKOUT|BUILD|TEST]-<module-name>-<configuration-name>.log
```

2. All packages are stored in the directory

```
dist
```

The packages are organised in a tree that mirrors the structure of the ETICS software repository:

```
<project-name>/<module-name>/<version>/<platform>/<package-name>
```

The packages are by default taken from the following locations in each module root:

(S)RPMS: <module-root>/RPMS

(src.)tar.gz: <module-root>/tgz
 dsc: <module-root>/debs
 deb: <module-root>/debs

The location where the publisher looks for packages can be changed by using the following built-in packager properties:

Table 19: publishing properties

Property name	Default value if not specified	Type	Description
package.tgzLocation	\${location}/tgz	string	The default location where generated tarballs are stored in each module at the end of a build and where the Publisher looks for them
package.SRPMSTLocation	\${location}/RPMS	string	The default location where generated source RPMS are stored in each module at the end of a build and where the Publisher looks for them
package.RPMSSTLocation	\${location}/RPMS	string	The default location where generated binary RPMS are stored in each module at the end of a build and where the Publisher looks for them
package.SDEBSSTLocation	\${location}/debs	string	The default location where generated source dsc packages are stored in each module at the end of a build and where the Publisher looks for them
package.DEBSSTLocation	\${location}/debs	string	The default location where generated binary deb packages are stored in each module at the end of a build and where the Publisher looks for

			them
--	--	--	------

9.5. Build Reports

The build reports are organised in a hierarchical tree with a top *index.html* page in the *reports* directory that shows summary information for the build and links to the detailed build reports, metrics, tests, etc. The exact same HTML report is available both for local and remote build and test. For more information about the reporting functionality refer to Chapter 11.2.

10. TESTING CONFIGURATIONS

10.1. Overview

Previous chapters described how to get a project into a workspace, how to browse its structure; edit it by creating, modifying or deleting modules, configurations and related commands, properties and dependencies. We also covered how to build configurations to produce build artefacts.

Once the information is registered in the ETICS system, we can use the ETICS client to execute tests, calculate metrics and generate reports. This chapter will describe the commands used to perform these operations and how to control the way tests are performed.

Testing configurations is generally used to perform *system tests*, as oppose to *unit tests* performed during the build procedure, as described in Chapter 9 - Building Configurations, as part of the *test* target. Each configuration can have *Test Commands* that include the commands that will be executed during a test procedure. As for the build procedure, before executing tests, a checkout is required. Therefore, in the following section, it is assumed that a project has been inserted in the workspace and that a suitable configuration has been checked out.

10.2. The etics-test Command

The ETICS command to perform builds is called `etics-test`. The syntax of the command is:

```
etics-test [options] [<module-name>]
```

The command has several options that allow controlling how tests are executed. The following table shows the most common options, while more advanced options will be described in the following sections:

Table 20: Options for `etics-test` command

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-c, --config <configuration-name></code>	Define a specific configuration to be used instead of the default one, where default is the first configuration found in the store. This is normally not required unless multiple or non-default configurations of the selected module have been checked out
<code>-p, --property <property=value></code>	This option allows passing properties to the test process. If the property is already defined, its value is overridden by the value specified in the command-line. To pass multiple properties, use multiple <code>-p</code> options
<code>-e, --env <env=value></code>	This option allows passing environment variables to the test process. If the variable is already defined, its value is overridden by the value specified in the command-line. To pass multiple variables, use multiple <code>-e</code> options
<code>-t, --target <target-name></code>	Execute the test stopping at the specified target. If not specified the publish target is executed. Allowed targets are: clean, init, test
<code>--platform <platform-name></code>	Overwrite the local platform (useful for testing or when the local platform is not a valid ETICS platform, but it's compatible with one).

<code>--notest</code>	Do not perform the actual test
<code>--continueonerror</code>	Do not stop testing if an error is found
<code>--nodeps</code>	Only test the currently specified module (do not test children and dependencies)
<code>--verbose</code>	Print verbose messages
<code>--version</code>	Return the current client version number.

10.2.1. How to Test a Configuration

The standard command to test a configuration is:

```
etics-test <module-name>
```

This command tests the default configuration of the specified module. The module must be a module within the module structure checked out with the `etics-checkout` command. The default configuration is the first configuration found in the local metadata store.

When run by itself, the command tests the current configuration of the current project. Therefore

```
etics-test ➔ etics-test <current-project>
```

This assumes that a valid project configuration has been checked out or the command will fail prompting to check out the project.

If more than one configuration of the same module exists in the store, for example because a non-default configuration has been checked out (see Chapter 6 for more information about checking out configurations), the `-c` option can be used to select a specific configuration:

```
etics-test -c <configuration-name> <module-name>
```

10.2.2. Test Targets

The test targets are defined by the commands available in the Test Commands set associated to a configuration for the current or the default platform. The available commands have been described Section 1.2.4:

Table 21: Test targets

Command name	Description	Mandatory
clean	Command to clean	No
init	Command to perform initialisation operations before compiling (e.g. deployment and configuration of services)	No
test	Command to execute the test	No

The `etics-test` command executes the target in the following order:

```
init ➔ test
```

stopping after the execution of the target specified with the `-t` option or after `test` if `-t` is not used. The `clean` target must be executed separately using the `-t` option.

If a target command is not specified in the set it is normally simply skipped.

10.3. Test Reports

The test reports are organised in a hierarchical tree with a top `index.html` page in the `reports` directory that shows summary information for the test and links to the detailed test reports, metrics, etc. The exact same HTML report is available both for local and remote build and test. For more information about the reporting functionality refer to Chapter 11.2 (“Submitting Builds and Tests Using the ”). The ETICS Publisher is also invoked during a test procedure, following a similar process as described in section 9.4 - The ETICS Publisher, but without the packaging.

11. SUBMITTING REMOTE BUILDS AND TESTS TO THE ETICS SYSTEM

This chapter describes two ways of submitting builds and tests. The user can either use the Build and Test Web Application (WA hereafter), or the command-line client.

11.1. Submitting Builds and Tests Using the Web Application

Besides modelling capabilities, the WA provides the user with remote build and test submission facilities. Similarly to in CLI-based client, the WA provides the user with the ability to submit remote builds and tests to a large number of host types and platforms. Following the execution of a remote build and test, a report is automatically generated and registered. If a build is issued, artefacts can be published to the ETICS repository. The ETICS remote builds and tests implementation leverage the Metronome infrastructure.

Remote build and test can be submitted through the WA once a specific configuration has been selected.

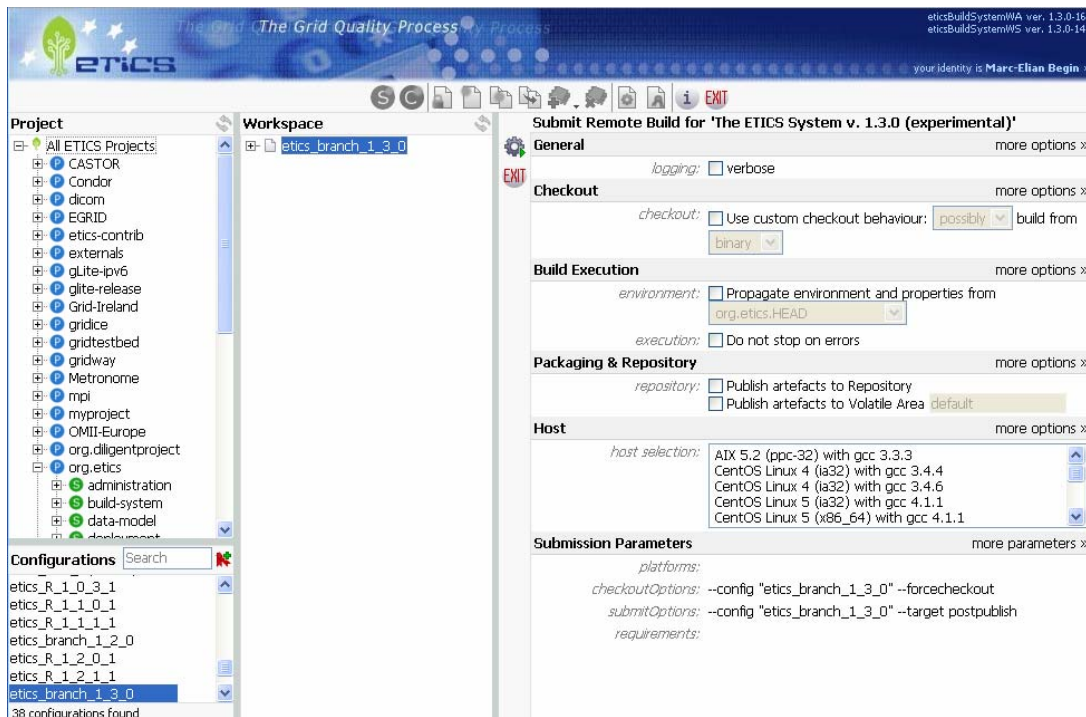


Figure 40 - Build Submission Page

Remote submission can also be triggered for more than one platform at the same time by selecting multiple entries in the 'host selection' list.

11.2. Submitting Builds and Tests Using the Command-Line Client

The ETICS command to submit build and test is called `etics-submit`. The syntax of the command is:

```
etics-submit <operation> [options] [<module-name>]
```

It supports two operations: *build* that submits a remote build job, and *test* that submits a remote test job.

The following table shows the options available for submitting a remote build or test:

Table 223: Options for remote build and test

Property name	Default value if not specified	Type	Description
--platforms	Current local platform	Comma separate list of platform strings (do not use blanks or quotes)	The list of platforms where to build and test
--project	Current project if defined		Define the name of the project to get for the build and test
-- project-config			Define a specific project configuration for the metadata
--fromsource			When possible, check out source code instead of downloading binaries.
--frombinary			When possible, download binaries instead of checking out source code. This implies that packages will be built (this is the default if not option is specified).
--fromsourceonly			Check out source code only. This means that all the configurations, including dependencies have to be available in source code form or the operation will fail.
--frombinaryonly			Checkout binaries only. This means that all the configurations, including dependencies have to be available in binary form, except the modules specified to the command.
--volatile <namespace>			Use a named volatile repository to look for packages. If a package

<p>--defaultvolatile</p>			<p>is not found in the volatile repository, it is searched for in the permanent repository, before giving up unless --volatileonly is used. This option cannot be used together with --defaultvolatile.</p> <p>Use the default volatile repository to look for packages. If a package is not found in the volatile repository, it is searched for in the permanent repository, before giving up unless --volatileonly is used. This is equivalent to --volatile=default. This option cannot be used together with --volatile.</p>
<p>--volatileonly</p>			<p>Use only the specified volatile repository to look for packages. If a package is not found in the volatile repository, it is not searched for in the permanent repository.</p>
<p>--register</p>			<p>Register the build packages and reports in the permanent ETICS repository (it requires permissions, ask your project administrator).</p> <p>Packages are always stored in a volatile repository, even if this option is not used.</p>
<p>--register-volatile <namespace></p>			<p>Register the build packages and reports in the named volatile repository instead of</p>

		the default one. If this option is not used the packages are stored in the default volatile repository.
--runasroot		Run the remote build and test as super user (e.g. root on linux/unix).
--freeze <time>		Freeze the remote node after execution of the build and test. The <time> parameter must be in the following format: '<number>m' for minutes '<number>h' for hours.
		WARNING: If this option is combined with --runasroot the machine will not be available for further submissions which might starve the resource pool.
--private-resource <private-resource-name>		Run the remote build and test on a private resource. In order for the match making to succeed, at least one machine must have been register with the key <private-resource-name>.
--requirements		Specify custom requirements for the match making of the remote resources.
-c, --config <configuration-name>		Define a specific configuration to be used instead of the default one, where default is the first configuration found in the store. This is normally not required

			unless multiple or non-default configurations of the selected module have been checked out
-p, --property <property=value>			This option allows passing properties to the test/build process. If the property is already defined, its value is overridden by the value specified in the command-line. To pass multiple properties, use multiple -p options
-e, --env <env=value>			This option allows passing environment variables to the test/build process. If the variable is already defined, its value is overridden by the value specified in the command-line. To pass multiple variables, use multiple -e options
-t, --target <target-name>			Execute the test/build stopping at the specified target. If not specified the publish target is executed. Allowed targets are: clean, init, test
--notest			Do not perform the actual test
--continueonerror			Do not stop testing/build if an error is found
--verbose			Print verbose messages
--version			Return the current client version number.
--nobuild			Do not perform the build, just print the sequence
--force			Force the build of

--urlname <report-name>			unmodified modules Specify a custom string to be used in the URL for the published build/test reports.
--shallowbindeps			When checking out using the --frombinary or --frombinaryonly options, dependencies of binary packages are not checked out.
--runtimedeps			When checking out configurations, dependencies of run-time dependencies are also processed. It is useful to make sure that the final list of packages contains everything needed to deploy the components.
--createsource			Create source tarball and RPMS
--nodistname			Do not append the distribution name to the RPM revision. It does not have effect if --userspec is used.
--usertimestamp			Append a timestamp to the RPM revision number. It does not have effect if --userspec is used.
--versioneddeps			Use version information when setting package dependencies in RPMS (e.g., etics-client >= 1.0.0). Cannot be together with --strictversioneddeps
--strictversioneddeps			Use strict version information when setting package dependencies in

--userspec		RPMS (e.g., etics-client = 1.0.0). Cannot be used together with --versioneddeps Use a user-defined spec file. The spec file must be placed in <moduleroot>/project. If this option is used all other packaging options are ignored.
------------	--	--

The URL for the published build/test reports takes the form:

```
http://<etics-servername>/rundir/<urlname>/reports
```

The standard URLs will be created even if the <report-name> is specified.

11.2.1. How to submit a remote build job

The standard command to submit a remote build job:

```
etics-submit build <module-name>
```

11.2.2. How to submit a remote test job

The standard command to submit a remote test job:

```
etics-submit test <module-name>
```

12. ACCOUNT INFORMATION

The ETICS command to retrieve and show account information is called `etics-log`. The syntax of the command is:

```
etics-log [options] [<module-name>]
```

The following table shows the options available for retrieving and showing account information:

Table 234: Options for account information

Property name	Default value if not specified	Type	Description
--project	Current project if defined		Define the name of the project to get for the build and test.
--dn <dn>			Specify the dn information of user's certificate.
--mydn			Select the certificate that belongs to the user running the command.
--operation <operation-type>	“		Specify the type of operation (add, modify, remove)
--component subsystem -s, --			Specify the type of a moduleName. Mandatory without having run etics-get-project
--startdate <start-date>			Specify the lower bound of the search range. Optional parameter. Set the date and time when the log information will start. If it is not set, there is not any lower-limit value.
--enddate <end-date>			Specify the lower bound of the search range. Optional parameter. Set the date and time when the log information will be stopped. If it is not set, there is not any upper-limit value.

--limit <number>			Specify the maximum number of returned entries.
-c, --config <configuration-name>			Define a specific configuration.
--verbose			Print verbose messages

The <start-date> parameter must be in one of the following formats:

the absolute start time:

'dd/mm/yyyy hh:mm'

assuming hh_mm to be 00:00

'dd/mm/yyyy'

The <end-date> parameter must be in one of the following two formats:

to stop after 123 minutes from <start-date>

'+123'

to stop after 321 hours from <start-date>

'+321'

or

the absolute stop time:

'dd/mm/yyyy hh:mm'

assuming hh:mm to be 23:59:

'dd/mm/yyyy'

13. REPOSITORY

As outcome of remote build and test submissions, the ETICS system produces different types of artefacts: packages generated during the build process, build and test reports, and metrics collected by the plug-in framework (see Chapter 16). The ETICS Repository is the standard location where all the software artefacts are stored and archived. In addition to the artefacts generated by the ETICS system, third party packages (e.g. externals) are also available as dependencies to build software.

The repository can be accessed and used in different ways:

1. The ETICS client communicates with the repository to get the packages during the checkout phase of a build or test procedure. Following the successful execution of a remote build, the ETICS Build and Test Web Service can store in the repository all the built packages (for local builds the built packages are only stored in the local workspace).
2. The user communicates with the repository via the Repository web application to download the packages built during previous remote builds, to analyse the reports generated, or the metrics collected.
3. The ETICS team adds to the repository as *externals*¹ the third party packages requested by the ETICS user community.

The repository is composed of two main parts: a set of volatile storage areas and a permanent storage area for the registration of packages together with related reports and metrics. These two parts have different purposes.

13.1. Volatile Storage AREAS

Volatile storage areas are temporary storage spaces that users can allocate on the repository for their private use. Volatile storage areas are used as locations where to find all packages, related reports and metrics that have been generated during the remote build operations on the ETICS infrastructure. These areas can be used by developers to check whether the generated packages are correct or not, to analyse the reports of builds or tests for debugging purposes, or to check the metrics collected by the client. They can also be used as an exchange point for certification. Unlike the *Registration storage area*, packages, reports and metrics that are registered in the *Volatile storage area*, overwrite existing ones if previously stored. The overwritten packages and reports are still available with other download locations to let the developer analyse the differences between consecutive versions but the artefacts considered by the Build and Test system are only the latest ones submitted (the old packages and reports are stored with a time stamp of when they are overwritten). By default all artefacts are registered in a default volatile area called *default*, unless a specific volatile area is specified by the user. If the user specifies a different volatile area, all generated packages will be registered in that area. New storage areas will be automatically created as specified by the user.

Issuing the following command, the module <module-name> will be remotely built and by default the packages will be registered in the *default* volatile area:

```
etics-submit build <module-name>
```

Since every `remote-build` artefact will be registered by default in the *default* area, artefacts already in this storage area may be unexpectedly overwritten. To avoid this, the repository provides custom volatile areas, where users can specify the name of the volatile area where the artefacts are to be registered. The following command executes a `remote-build` and stores the artefacts in the specified volatile area:

```
etics-submit build --register-volatile <area-name> <module-name>
```

¹ The *externals* project in ETICS contains all third party modules that are not necessarily built using ETICS, but used as dependencies by other projects.

If `<area-name>` is not present in the repository, it will be created as part of the artefact registration. Be aware however that the volatile storage area is scratched on a regular basis, according to the available space on our disk server.

Once a volatile storage area is present in the repository, it is possible to set it as the location where to get the binaries during the checkout operation. If the package is not present in that area, by default the client will look for the package in the *permanent repository*. It is important to understand that only the latest version of stored packages is available in volatile areas for the checkout operation. The following command specifies a specific volatile area for checkout:

```
etics-checkout --volatile <area-name> <module-name>
```

for a remote build:

```
etics-submit build --remote-volatile <area-name> <module-name>
```

or for a remote test:

```
etics-submit test --remote-volatile <area-name> <module-name>
```

13.2. Registered Storage Area

The registration storage area is the official location where to store released packages together with their related reports and metrics. The registered storage is the location to chose in order to save permanently packages (e.g. when a package can be made public, final version of a configuration). Packages stored in the permanent storage area cannot be overwritten with other packages. Build reports and their metrics are also stored together with their packages. In other words, if the build doesn't generate any package or it generates only packages that are already present in the registered area, the report and the metrics will not be saved in the registered part of the repository. Test reports and their metrics are on the other hand always registered. This area is set by default as the location where binaries are downloaded from during the checkout operations.

To register the artefacts in the permanent storage area of the repository, use the `--register` option. The user must also have *Developer* role on the corresponding configuration.

```
etics-submit build --register <component-name>
```

Note that by default the `etics-submit` command doesn't register artefacts.

13.3. Repository Web Application

The ETICS repository provides a web application to browse both *Volatile* and *Registered* storage areas of the repository. It provides an easy way to access generated reports, packages and metrics. This web application is included in the ETICS Portal as panel at the following URL:

<https://etics.cern.ch/eticsPortal> (panel Repository)

The Repository web application is composed of two parts: a tree on the left showing the hierarchical structure of the repository and a panel on the right showing information on the selected node of the tree.

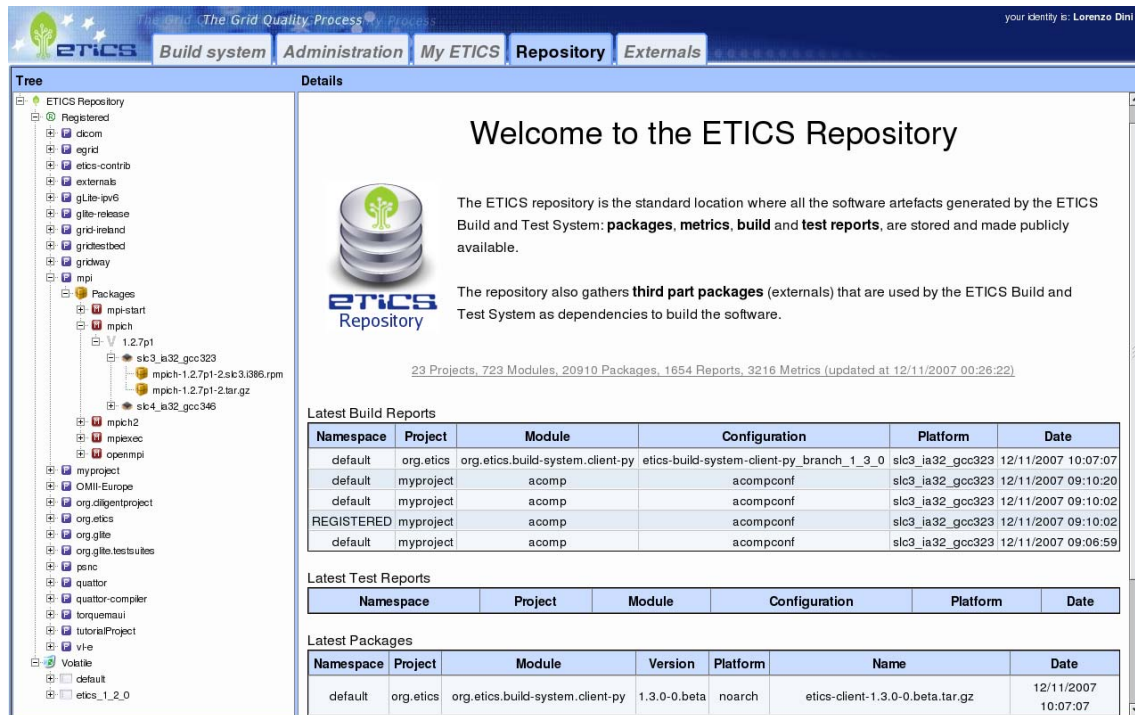


Figure 41: Repository web application

The Tree Panel

On the left of the Repository web application, a tree panel representing the hierarchical structure of the repository is available to access all artefacts.

On the top of the tree, a tool bar is available with three buttons to refresh the currently selected node of the tree and to resize the panel moving the split left or right.

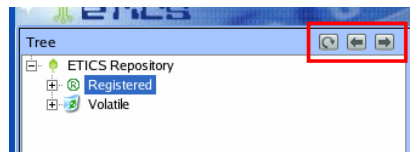


Figure 42: Tree tool bar

The top of the tree is divided in *Registered* and *Volatile* nodes. The first node is the permanent storage area, the second node includes a list of volatiles areas created by users and the *default* volatile area.

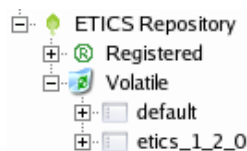


Figure 43: Registered and Volatile repository nodes

Inside each area a list of project is present. For each project, it is possible to access the information in two ways: *by packages* or *by reports*.

Browsing a project *by packages* allows the user to focus on the outcome of the build or test and choose a particular version of a module selecting version, platform and package. Once a package is identified,

it is possible to download it, to check its contents if it is one of the supported formats, to see the related build information to verify where the package comes from and who has generated it, or finally to view the associated report to check in details the generation process and the environment on which the package has been built. This view of the repository addresses the need of downloading specific versions of software also verifying their origin (usually used by certification and integration teams, and by end users).

Browsing a project *by reports* instead, provides a way to access the same information focusing on the generation process and not on the final outcome of a submission. Selecting a configuration and a platform allows users to browse all builds and tests performed on the system and to view all reports that have been generated during these submissions. Checking the reports, it is possible to see what packages have been generated and to download them. This view of the repository addresses the need to keep track of every submission and check their results (usually used by developers and testers).

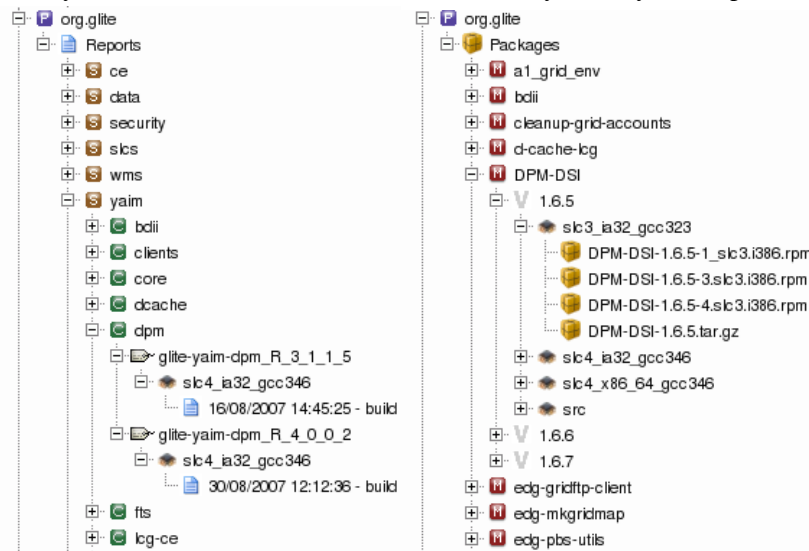


Figure 44: Browsing a project by packages or by reports

As said before, in the volatile areas or the repository, not only the latest packages, but also the versions that have been overwritten are stored. The old versions are renamed appending the time stamp of when they have been created providing the history of a package. These packages are available until the cleaning mechanism of the repository deletes them after a certain amount of time or after a certain number of consecutive builds.



Figure 45: Browsing latest and overwritten packages

Reports are shown ordered by date starting from the latest. Two types of reports are displayed with different icons: the main reports and the module reports.

The *main report* is shown where the selected configuration was directly built or tested. This means that the user actually submitted that configuration for a build or test. For these configurations, the report shown is the global report on which all information of the submission are shown in a summary and all the other sections are available through links from the main page. This report is identified with a full page icon.

The *module report* is shown if the selected configuration was not directly submitted by the user, but was triggered as a direct or indirect dependency or child of a configuration submitted by the user. For these configurations, the report shown is the log of the execution of that module instead of the whole report of the submission. This report is identified with a half page icon.

Using links, it is always possible to jump from a module report to the summary, or to drill down from a summary to module reports.

Generally, when browsing a platform of a certain configuration, a list of reports will be available. Some of them will be main reports, because the user (typically the developer) submitted a build or a test on that specific configuration, some other will be module reports because that configuration was triggered by another configuration through dependency or child relationships. In both cases it is possible to keep track of the build or test information.

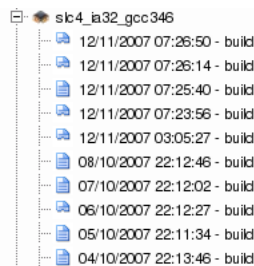


Figure 46: Browsing *main* and *module* reports

Statistics and Latest Panel

Every summary panel shown on the right side of the Repository web application contains some statistics and latest artefact tables where it is possible to find information about the contents of that particular branch of the repository and the latest reports and packages submitted for that branch.

The statistics are refreshed once per day by the first request of that day.

Every latest table contains a refresh button to re-query the repository for latest artefacts.

23 Projects, 723 Modules, 20910 Packages, 1654 Reports, 3216 Metrics (updated at 12/11/2007 00:26:22)

Latest Build Reports

Namespace	Project	Module	Configuration	Platform	Date
default	org.etics	org.etics.build-system.client-py	etics-build-system-client-py_branch_1_3_0	slc3_ia32_gcc323	12/11/2007 10:07:07
default	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:10:20
default	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:10:02
REGISTERED	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:10:02
default	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:06:59

Latest Test Reports

Namespace	Project	Module	Configuration	Platform	Date
default	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:10:02
REGISTERED	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:10:02
default	myproject	acom	acompcnf	slc3_ia32_gcc323	12/11/2007 09:06:59

Latest Packages

Namespace	Project	Module	Version	Platform	Name	Date
default	org.etics	org.etics.build-system.client-py	1.3.0-0.beta	noarch	etics-client-1.3.0-0.beta.tar.gz	12/11/2007 10:07:07
default	org.etics	org.etics.build-system.client-py	1.3.0-0.beta	noarch	etics-client-1.3.0-0.beta.py2.2.noarch.rpm	12/11/2007 10:07:07
default	org.etics	org.etics.build-system.webservice	1.3.0-0	noarch	etics-webservice-1.3.0-0.tar.gz	09/11/2007 14:48:02
default	org.etics	org.etics.build-system.webservice	1.3.0-0	noarch	etics-webservice-1.3.0-0.noarch.rpm	09/11/2007 14:48:02
default	org.etics	org.etics.nmi.scripts	0.0.0-0	noarch	etics-nmi-scripts-0.0.0-0.tar.gz	09/11/2007 13:06:55

Figure 47: Summary panel

Clicking on one of the latest report opens a pop-up with the selected report. The user can also download one of the latest packages by clicking on the corresponding entry.

14. ANALYSING BUILD AND TEST REPORTS

During the build and test process, locally and remotely, the ETICS client assembles a rich report including information such as which commands were used to execute the procedure, high-level information regarding which module and configuration are being built and/or tested, logs generated during the process and reports generated by the plug-in system (see Chapter 15.2: Administration with the Command-Line Client). These build and test reports are accessible via the Report Browser web application (see section 13.3 “Repository Web Application” for details) or via the CLI commands `etics-get-report` and `etics-get-rundir`.

14.1. Standard Reports

Each build and test procedure generates a rich report which can be viewed locally in the workspace ‘*reports*’ directory. Further, this report is also available for remote build and test submissions.

Each top level report contains the following information:

- **Module Summary:** Information on the module that has been built or tested, including the name of the project, module name, description and version, release, vendor and licence type.
- **Execution Summary:** Information on the build process, including final results of the build and test, number and percentage of successfully built components, start time, end time and duration of the process, configuration used, tag on the VCS, path of the VCS root and the platform on which the process has been executed.
- **ETICS Commands:** The list of ETICS commands executed on the worker node.
- **Environment Properties:** The list of environment variables used to execute the build and test.



Figure 48: Summary build and test page

On the top left part of the report summary, a side menu is present. From here are accessible the logs, the package lists report, metrics pages, and if available test reports and custom reports. By clicking on the *Logs* link, the user can access the logs section.

The logs page shows summary information of the configuration built or tested and the list of all its sub-configurations and dependencies – i.e. these entries are ordered, following the same order used during the build and test procedure, with the requested module last, since all its children and dependencies are executed first.

From the list it is possible to see the component name, the configuration name, the last build time and the result of the build.


ETICS
Build System

Project name: org.glite

Project config: glite_branch_3_1_0

Module name: org.glite.rgma

Module config: glite-rgma_R_5_0_60

Build start time: 18/02/2007 20:09:16

Success rate: 100% (52/52)

Status: Success

Page generated at 18/02/2007 20:33:03

[Back to module overview page](#)

Component name	Configuration name	Last build time	Result
jdk	jdk v. 1.5.0_06	18/02/2007 20:09:24	Success
ant	ant v. 1.6.2	18/02/2007 20:09:26	Success
eggee-ant-ext	eggee-ant-ext v. 0.4.0	18/02/2007 20:09:27	Success
jalopy	jalopy v. 1.0b10	18/02/2007 20:09:28	Success
checkstyle	checkstyle v. 3.5	18/02/2007 20:09:29	Success
ant-contrib	ant-contrib v. 1.0b1	18/02/2007 20:09:31	Success
cpptasks	cpptasks v. 1.0b2	18/02/2007 20:09:32	Success
log4j	log4j v. 1.2.8	18/02/2007 20:09:33	Success
junit	junit v. 3.8.1	18/02/2007 20:09:34	Success
org.glite.build.common-java	glite-build-common-java_branch_3_1_0	18/02/2007 20:09:37	Success
tomcat	tomcat v. 5.0.28	18/02/2007 20:09:40	Success
globus	globus v. 4.0.3-VDT-1.6.0	18/02/2007 20:09:46	Success
mysql-devel	mysql-devel v. 4.1.11	18/02/2007 20:09:50	Success
expat	expat v. 1.95.7	18/02/2007 20:09:53	Success
bcprov-jdk14	bcprov-jdk14 v. 1.22	18/02/2007 20:09:54	Success
cppunit	cppunit v. 1.10.2	18/02/2007 20:09:56	Success
org.glite.security.voms	glite-security-voms_R_1_7_14_2	18/02/2007 20:09:58	Success
org.glite.security.voms-clients	glite-security-voms-clients_R_1_7_14_2	18/02/2007 20:10:01	Success
org.glite.security.build	org.glite.security.build_R_3_1_33_1	18/02/2007 20:10:03	Success
org.glite.security.test-utils	glite-security-test-utils_R_1_6_0	18/02/2007 20:21:06	Success
axis	axis v. 1.1	18/02/2007 20:21:08	Success
org.glite.security.util-java	glite-security-util-java_R_1_3_8_1	18/02/2007 20:21:11	Success

Figure 49: List of built and tested modules

Clicking on any module displayed in the module list (see Figure 49), detailed logs are available. There, a detailed page (see Figure 50) is available containing the logs generated during the checkout or download operations, then logs generated during the build and finally logs generated during the test operations.

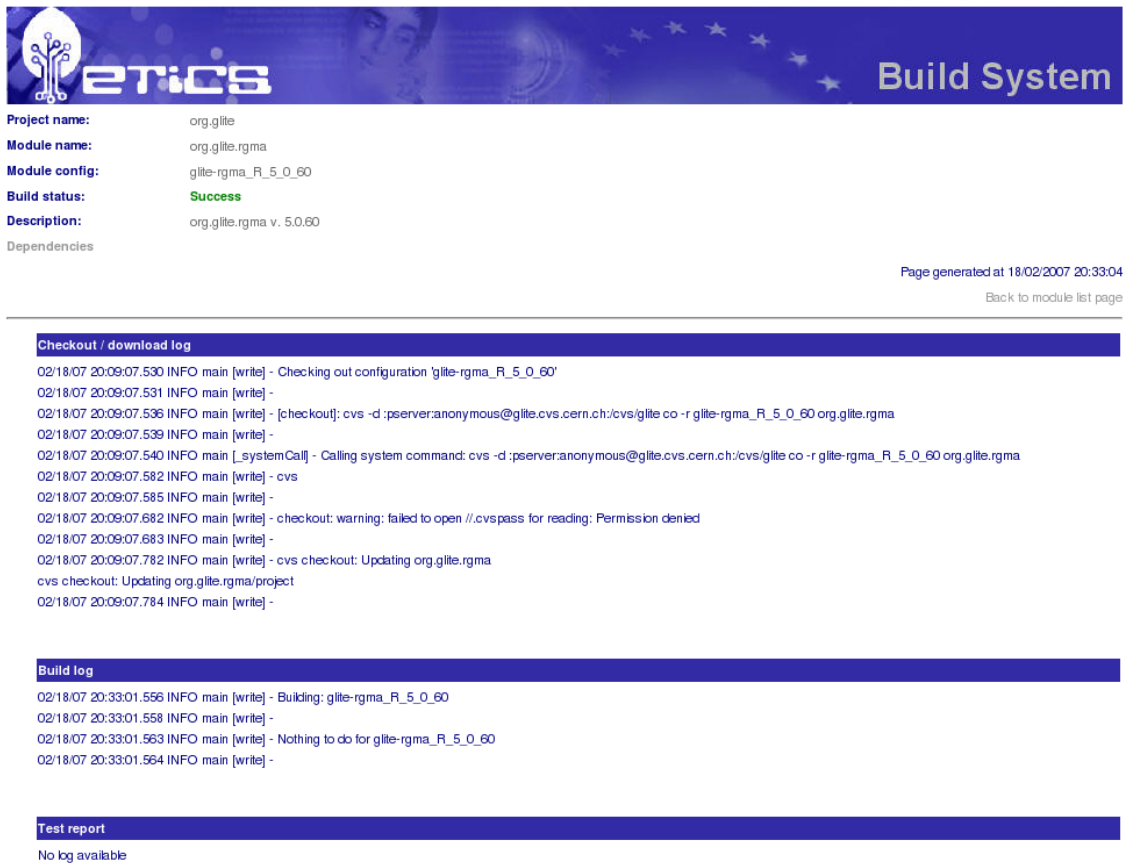


Figure 50: Detailed log

To access the dependency resolution, a *Dependencies* link is present.

The dependency resolution is composed of two lists:

- **Dependencies:** a list of dependencies for the current module.
- **Used by:** a list of the components that depend on the current module.

For each entry, the scope of the dependency is visible: **B** are build type dependencies, **R** are runtime dependencies, **BR** are build and run time dependencies.

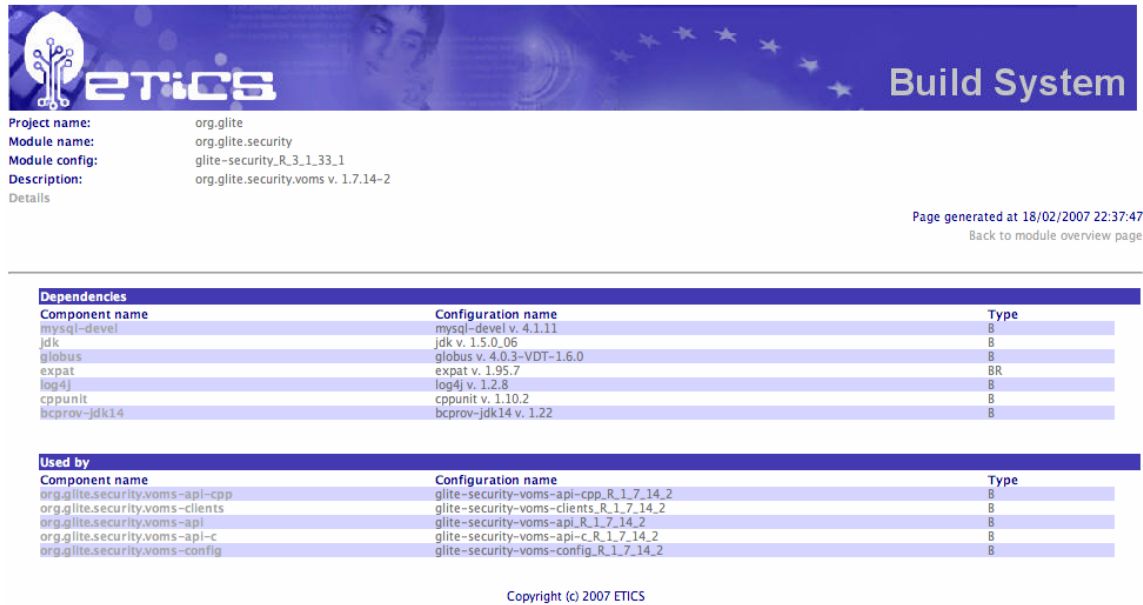


Figure 51: Used-by and dependencies page

The *Package List report* provides an overview of the different packages created or used during a build as build or run-time dependencies. The report by default shows all packages, but the user can also select specific package types (such as rpms or tar.gz files). The report displays the package file name, the description, the package type and a status flag. If the package is available in the repository the displayed flag is 'Ok', otherwise the flag is 'Not found'. The package file names have links to download the corresponding files from the repository. Additionally, a link at the top of the report provides a downloadable text file of the package list that can be used directly to download all packages using for example standard wget commands (Figure 52).



ETICS

Build System

Project name:

org.etics

Project config:

Unknown

Module name:

org.etics

Module config:

etics_R_1_0_1_1

Page generated at 12/06/2007 16:27:27
[Back to module overview page](#)

Package type:

☒ All ☐ rpm ☐ tar.gz

[Download as text](#)

Package name	Description	Type	Status
4Suite-1.0.2-1.slc4_py2.3.i386.rpm	4Suite is a library of integrated tools (including convenient command-line tools) for XML processing, implementing open technologies such as DOM, RDF, XSLT, XInclude, XPointer, XLink, XPath, XUpdate, RELAX NG, and XML/SGML Catalogs	rpm	Ok
4Suite-1.0.2-1.tar.gz	4Suite is a library of integrated tools (including convenient command-line tools) for XML processing, implementing open technologies such as DOM, RDF, XSLT, XInclude, XPointer, XLink, XPath, XUpdate, RELAX NG, and XML/SGML Catalogs	tar.gz	Ok
Metronome-2.2.6-0.noarch.rpm	The NMI Build & Test software.	rpm	Not found
Metronome-2.2.6-0.tar.gz	The NMI Build & Test software.	tar.gz	Not found
MySQL-client-standard-5.0.27-0.slc4.i386.rpm	MySQL is a client/server implementation consisting of a server daemon (mysqld) and many different client programs and libraries. The base package contains the MySQL client programs, the client shared libraries, and generic MySQL files	rpm	Not found
MySQL-client-standard-5.0.27-0.tar.gz	MySQL is a client/server implementation consisting of a server daemon (mysqld) and many different client programs and libraries. The base package contains the MySQL client programs, the client shared libraries, and generic MySQL files	tar.gz	Ok
MySQL-server-standard-5.0.27-0.slc4.i386.rpm	MySQL server - the most popular open source database.	rpm	Not found
MySQL-server-standard-5.0.27-	MySQL server - the most popular open source database.	tar.gz	Ok

Figure 52: Package List report

The last section of the build and test reports is the metrics page. Clicking in the summary page on the *Metrics* link displays the list of high-level metrics (see Figure 53).

14.2. The etics-get-report command

The command `etics-get-report` downloads the logs section of the report browser for remote build and test reports and saves it in plain-text format. Its syntax is:

```
etics-get-report [options] [<gid>]
```

Where `<gid>` is the *Submission ID* returned by the `etics-submit` command (ignored if `--dn` option is set).

The following table shows the options available for the command:

Table 24: Options for `etics-get-report`

Option name	Default value if not specified	Description
<code>--dn <dn></code>	If both <code><gid></code> and <code>--dn</code> are omitted, all reports for jobs of the user running the command are saved	Save the reports for all the jobs submitted by the specified user. If this option is specified, <code><gid></code> value is ignored.
<code>--diff <other-gid></code>	No comparison will take place	Save also the reports for the job with id <code><other-gid></code> and compares them with the reports of <code><gid></code> , in a way similar to the <code>diff</code> command. Ignored if <code>--dn</code> is also specified.
<code>-l, --limit <number></code>	No limit	When getting the reports for all of a user's jobs, save only the last <code><number></code> jobs

<code>--verbose</code>	submitted. Ignored when <code><gid></code> is specified and <code>--dn</code> is not specified. Print verbose messages
------------------------	---

The plain-text reports are saved in the *remote/reports* directory of the workspace, one text file for every platform the job was submitted on. The report for a platform won't be saved if the job is still running or it wasn't submitted (i.e. `etics-status` command reports the status as *Running* or *Not found* for that job)

14.3. The `etics-get-rundir` command

The command `etics-get-rundir` saves locally all the rundirs generated by a remote submission. A *rundir* is a directory created on the build and test system web server, containing all the files needed to execute a submission on a single platform and the files generated by the job itself (including the report browser pages).

The command syntax is:

```
etics-get-report [options] [<gid>]
```

Where `<gid>` is the *Submission ID* returned by the `etics-submit` command (ignored if `--dn` option is set).

The following table shows the options available for the command:

Table 25: Options for `etics-get-rundir`

Option name	Default value if not specified	Description
<code>--dn <dn></code>	If both <code><gid></code> and <code>--dn</code> are omitted, all rundirs for jobs of the user running the command are saved	Save the rundirs for all the jobs submitted by the specified user. If this option is specified, <code><gid></code> value is ignored.
<code>-l, --limit <number></code>	No limit	When getting the rundirs for all of a user's jobs, save only the last <code><number></code> jobs submitted. Ignored when <code><gid></code> is specified and <code>--dn</code> is not specified.
<code>--verbose</code>		Print verbose messages

The rundirs are saved in the *remote/rundirs* directory of the workspace, one folder for every platform the job was submitted on. If the command is run multiple times for the same submission, only files added/changed since the last run will be downloaded. If the job is still running or it wasn't submitted (i.e. `etics-status` command reports the status as *Running* or *Not found* for that job) the command will issue a warning and download available files.

14.4. Specific reports

Depending on the type of build and test procedure, specific reports are generated. Most of these specific pages will be generated by plugins, either provided out-of-the-box by ETICS or custom plugins provided by users. For more details on plugins and the framework they run under, refer to Chapter 15.2 ("Administration with the Command-Line Client").

Single Lines Of Code Count report

The SLOCCount stands for *Single Lines Of Code Count*. The metrics summary page (see Figure 53) displays the high-level metrics for a build and test procedure, including for standard builds the SLOCCount. This metrics counts the number of lines of code, organised by language. The metrics list includes all the language present in the source code with the number of lines of code calculated for each.

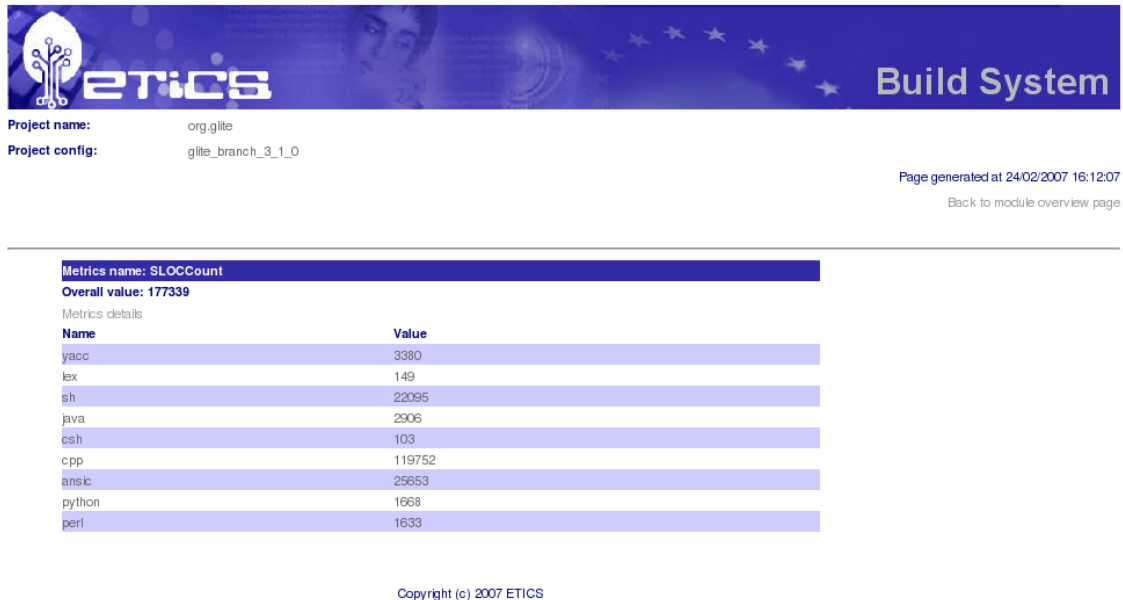


Figure 53: Metrics summary page

Clicking on *Metrics details*, a detailed graph is displayed (see Figure 54) with the metrics evaluated for every component part of the current report.

Single Lines of Code (SLOC) Summary

Project: glite_branch_3_1_0 (org.glite)
Configuration: glite-wms_R_3_1_26_1 (org.glite.wms)
Date: 24/02/2007 16:12:04

Total Physical Source Lines of Code (SLOC)

SLOC = 177339

Total SLOC grouped by language (dominant language first)

Language	Total SLOC
cpp	119752 (67%)
ansic	25653 (14%)
sh	22095 (12%)
yacc	3380 (1%)
java	2906 (1%)
python	1668 (0%)
perl	1633 (0%)
lex	149 (0%)
csh	103 (0%)

SLOC by language for all modules

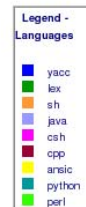
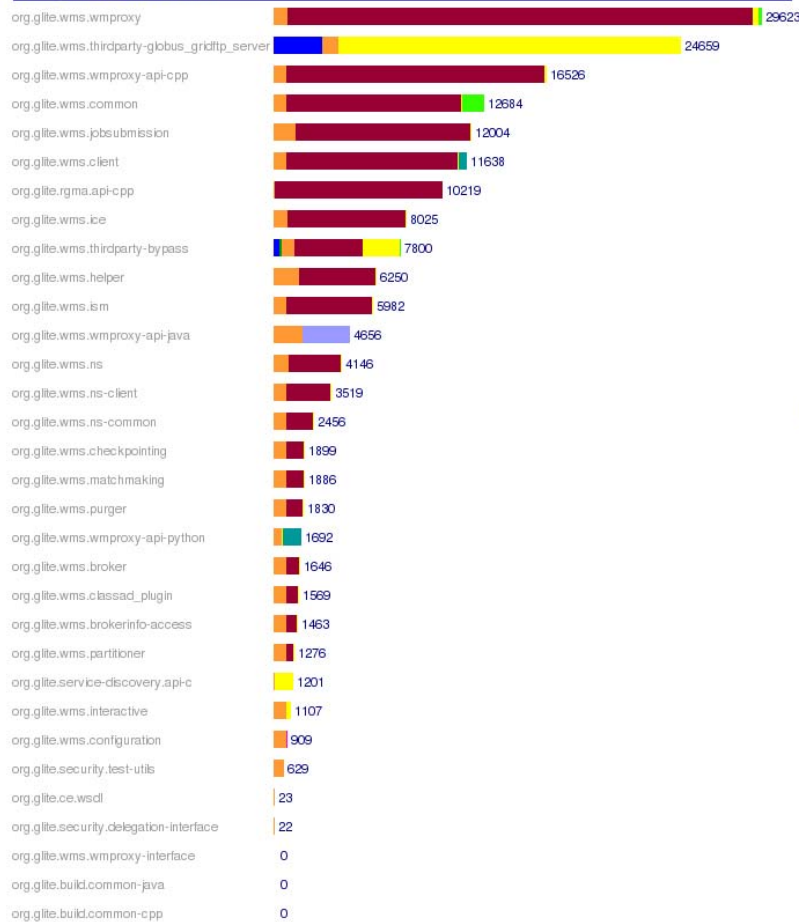
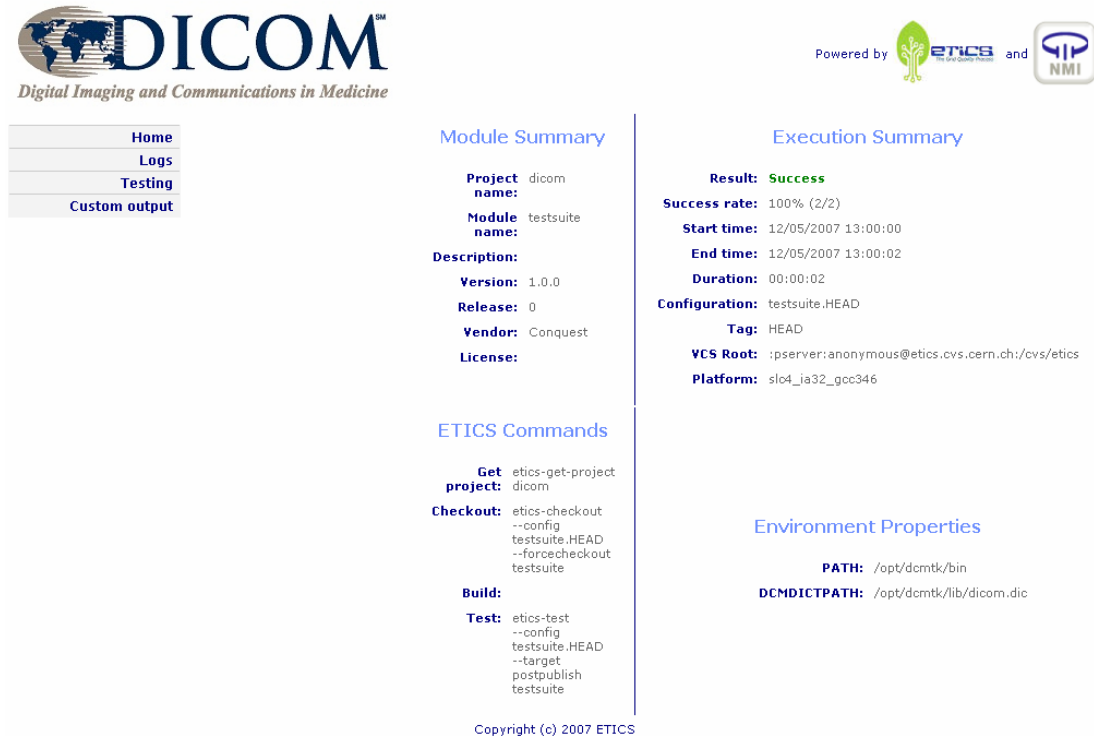


Figure 54: SLOC count metrics detailed page



Test and Custom reports

Test reports, either for unit or system tests, if generated, can be accessed from the report summary page. The same is true for custom reports.

A set of simple naming conventions are used to influence the generation of the summary page. If a file named *index-tests.html* is present in the *reports* directory of the workspace, an extra link “Testing” is added to list of links at the top left of the summary page.



DICOM
Digital Imaging and Communications in Medicine

Powered by  **ETICS** and 

Home
Logs
Testing
Custom output

Module Summary

Project name: dicom
Module name: testsuite
Description:
Version: 1.0.0
Release: 0
Vendor: Conquest
License:

ETICS Commands

Get project: etics-get-project dicom
Checkout: etics-checkout --config testsuite.HEAD --forcecheckout testsuite
Build:
Test: etics-test --config testsuite.HEAD --target postpublish testsuite

Execution Summary

Result: **Success**
Success rate: 100% (2/2)
Start time: 12/05/2007 13:00:00
End time: 12/05/2007 13:00:02
Duration: 00:00:02
Configuration: testsuite.HEAD
Tag: HEAD
VCS Root: pserver:anonymous@etics.cvs.cern.ch:/cvs/etics
Platform: sl04_ia32_gcc346

Environment Properties

PATH: /opt/dcmtdk/bin
DCMDICTPATH: /opt/dcmtdk/lib/dicom.dic

Copyright (c) 2007 ETICS

Figure 55: Summary report with testing and custom links

Similarly, if a file called *index-custom.html* is available in the *reports* directory, a supplementary link called “Custom output” is added to the summary report. Examples of both above mention cases can be seen in Figure 55.

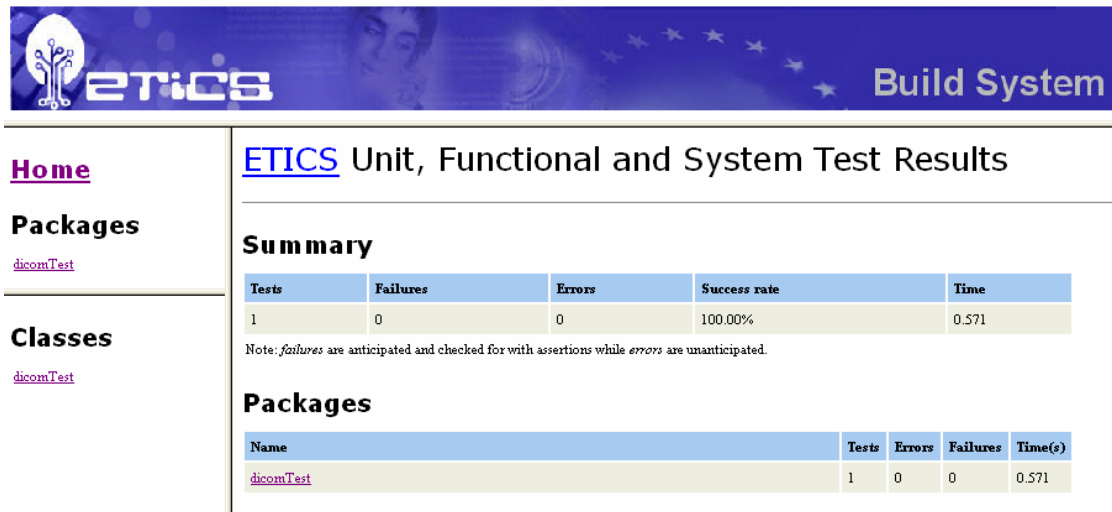


Figure 56: Example of detailed test report

Figure 56 shows an example of test report automatically generated by ETICS. Setting the *profile* configuration parameter to 'python', ETICS automatically executed all PyUnit¹ test present in the module and generates this report.

If for example a testsuite generates its own HTML custom report (see Figure 57), as described above, the report can be accessed from the summary page via the “Custom output” link.

¹ Similar support is been added for Java.



DICOM Image Conversion Results



MR-MONO2-16-knee.jpg

Figure 57: Example of custom report

This extensibility mechanism allows users to extend the generation of build and test reports, thus providing further information to facilitate the analysis of a build and test procedure.

15. USER AND MODULE ADMINISTRATION

Administration of users and modules can be performed using a dedicated web application or the command-line client. This chapter describes how to use both tools to perform administration tasks.

15.1. The ETICS Administration Application

The ETICS Administration application is a web-based application dedicated to manage projects and users in the ETICS system. This application is restricted to user with the **ETICS system administrator** and **ETICS module administrator** roles.

The ETICS system administrators may perform all available operations on any project, subsystem, component, configuration, role or user.

The ETICS module administrator can only administer project, subsystem, module or configurations for which he/she has this role on. In particular this means that he/she:

1. cannot edit users or add projects;
2. cannot add/remove role to elements outside his/her scope. For instance, a module administrator of a single subsystem cannot assign roles to other subsystems in the same project;
3. for every user, can add/remove any role to elements within his/her scope. For instance, a module administrator of a given subsystem can assign the role “*Module Administrator*” to any user on any component or configuration in this subsystem (including the subsystem itself).

The same user can be an ETICS module administrator for several components.

The following preconditions have to be satisfied before running the ETICS Administration application:

1. A valid user certificate is installed in a web-browser.
2. The user has the role *ETICS system administrator* and/or *ETICS module administrator*.

For details on setting-up certificates in your browser, refer to section 3.3 (“Enabling Security”).

To start the application, open the main page in a web browser – e.g.:

<https://etics.cern.ch/eticsPortal> (panel Administration)

15.1.1. Layout

The application screen is divided into 3 parts (see Figure 58 below):

1. The ETICS banner (top).
2. The menu (left side of the screen).
Functions selected in the menu are displayed in the workspace area. This menu only shows the functions available for the current user – for instance the option “*Add New Project*” is only visible for the *ETICS system administrator* (not the *ETICS module administrators*).
3. The workspace (right side of the screen).
This area displays the function selected from the menu. This is the only part of the application that changes content during the interaction, all next screenshots in this section will include only the workspace content.

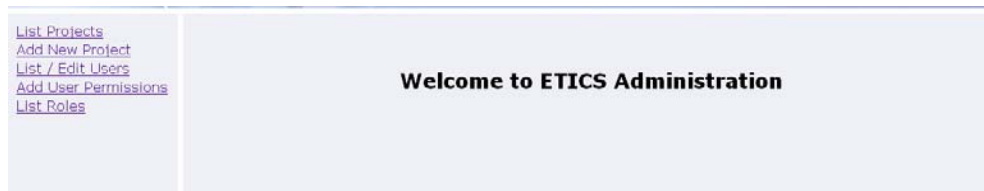


Figure 58: Administration web application

15.1.2. Manage projects

List projects

To list information related to projects, select option “**List Projects**” from the menu.


Project List	
	org.etics
The ETICS System	
Display Name:	ETICS System
ID:	69b6ab6a-f49f-48ac-becc-237e3f7a8644
Repository:	http://eticssoft.web.cern.ch/eticssoft/repository/
VCSRoot:	:pserver:anonymous@etics.cvs.cern.ch:/cvs/etics
Create date:	Wed Mar 22 08:51:42 CET 2006
Modify date:	Mon Sep 25 00:33:09 CEST 2006
Vendor:	ETICS
Homepage:	http://www.eu-etics.org
Users:	List user permissions for this project
	org.glite
gLite is the next generation middleware for grid computing developed by the EGEE project	
Display Name:	gLite Middleware

Figure 59: Project list

The project list is filtered depending on the user role:

1. All projects for the **ETICS system administrator**.
2. Only related projects for the **ETICS module administrator**.
For instance module administrator of subsystem “*build-system*” from “*ETICS System*” will see only one project – “*ETICS System*”.

Add a new project (system administrators only)

Only users with the role **ETICS system administrators** have access to this feature.

To add a new project to the ETICS system, select the option “**Add New Project**” from the menu.

Add New Project

Logo URL:	softweb.cem.ch/eticssoft/repository/org.etics/logo.jpg
Name:	org.etics
DisplayName:	ETICS System
Description:	
Repository:	http://eticssoftweb.cem.ch/eticssoft/repository/
VCSRoot:	pserver.anonymous@etics.cvs.cem.ch/cvs/etics
Vendor:	ETICS
Homepage URL:	http://www.eu-etics.org
<input type="button" value="Add Project"/>	

Figure 60: Add new project form

The following parameters can be specified:

Table 26: New project parameters

Parameters	Description
Logo URL	HTTP address to a graphics file with project logo (any public graphic file recognized by the web-browser, like gif, jpg, png, etc.).
Name	Project name.
DisplayName	Display name (more descriptive than name).
Description	Project description.
Repository	Binary repository root.
VCSRoot	VCS server address.
Vendor	Vendor name.
Homepage URL	Project home page.

Only the field “Name” is required, all others are optional.

15.1.3. Manage users

List users

To list users registered in the ETICS system, select the option “*List / Edit Users*” from the menu.

User List

Name: <i>karveeg Dagfinn</i> Email: <i>dagfinn.karveeg@fict.no</i> ID: <i>7c426440-8a8e-4330-87c9-b446c346888</i>	CN=Dagfinn Karveeg OU=org.et OU=etings@etings.org.et OU=etings@et O=etings	ACTIVE Permissions Edit User
Name: <i>Agustio Sanchez Carlos</i> Email: <i>Carlos.Agustio.Sanchez@cem.ch</i> ID: <i>c58c217f-c05e-4884-a652-4234ab72ac36</i>	CN=Carlos Agustio Sanchez CN=c58c217 CN=agustio OU=cem OU=Organic Units DC=cem O=cem	ACTIVE Permissions Edit User
Name: <i>Attili Fust</i>	Email: <i>attili.fust@cem.ch</i>	ACTIVE

Figure 61: User list

The following data is shown for every listed user:

1. User name, e-mail address, internal user ID.
2. Certificate name.
3. Activation status (active/inactive), link to view user's permissions, link to edit user data (only when the application user is the ETICS system administrator).

The user list is filtered depending on the application user role:

1. All users for the **ETICS system administrator**.
2. Only active users for the **ETICS module administrator**.

Edit user (system administrators only)

Only users with the role **ETICS system administrators** have access to this feature.

To edit information regarding registered ETICS user:

1. Select the option "**List / Edit Users**" from the menu.
2. Locate the user to edit (web-browser search functionality can be used).
3. Select the link "**Edit User**".

The following user data can be modified: first name, last name, e-mail address, certificate name, activation status.

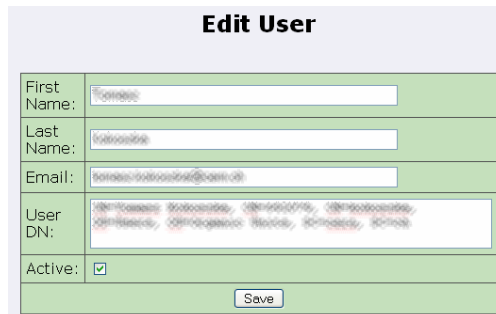


Figure 62: Edit user form

15.1.4. Manage user permissions

Permissions in the ETICS system are relations between users, ETICS modules/configurations and roles.

Permissions are propagated top-down in the ETICS hierarchy. This means that a user having permission A for an element X automatically has the same permission for all children of element X.

For instance, if user A has the role "*Module Administrator*" for a given subsystem in a given project, this means that user A has the same role for all configurations, components and components configurations children of this subsystem. However, as permissions propagate top-down, user A does not inherit the same role for other subsystems in the project.

There is a privileged role called **administrator** with an ETICS system-wide scope. Users having ETICS system administrator role are refer to as the **ETICS system administrators**.

User permissions can be managed only by the ETICS system administrator or the ETICS module administrator. The **ETICS system administrator** can manage permissions of all registered users for

all ETICS elements. The **ETICS module administrator** can manage permissions of all registered users but only in the scope of the ETICS element(s) he/she administrates.

For instance, if user X is an ETICS module administrator of a given subsystem in a given project, then he/she can only manage the permissions related to ETICS elements within this subsystem (including the subsystem itself).

List roles

To list all available roles predefined in the ETICS system, select the option “**List Roles**” from the menu. This list is static and cannot be modified.

Role List		
Name	Display Name	Description
Administrator	Administrator	This user can perform all the operations on the system including security-related operations
Developer List users with this role	Developer	This user has read/write access to configurations and related objects and properties
Guest List users with this role	Guest	This user can only get read-only access
Integrator List users with this role	Integrator	This user can submit builds and tests jobs
ModuleAdministrator List users with this role	ModuleAdministrator	This user can create and manage components and subsystems
ReleaseManager List users with this role	ReleaseManager	This user can tag artifacts and releases and upload them to the repository
Tester List users with this role	Tester	This user can submit tests and upload test results to the repository

Figure 63: Role list

Add user permission

To add a new role to a user, select the option “**Add User Permissions**” from the menu.

Add User Permissions	
Project:	* SELECT A PROJECT FROM THE LIST
Subsystem:	----
Component:	----
Configuration:	----
User:	* SELECT USER FROM THE LIST
Roles:	<div>NO ROLE SELECTED</div> <div>NO ROLE SELECTED</div> <div>NO ROLE SELECTED</div> <div>NO ROLE SELECTED</div> <div>NO ROLE SELECTED</div>
Add User Permissions	

Figure 64: Add user permissions form

The following parameters have to be specified:

Table 27: New user permission parameters

Parameters	Description
ETICS element	One of project, subsystem, component or configuration. The permission is assigned to the lowest selected element. For instance after selecting a project and a subsystem, the permission will be assigned to the subsystem (not the project). The ETICS system administrator may specify any element.

User	The ETICS module administrator only sees elements within his/her scope.
Roles	Any registered ETICS user. Any ETICS role. Several roles might be assigned at the same time (at least one).

List user permissions

The **ETICS system administrator** can view all the permissions.

The **ETICS module administrator** sees only permissions related to his/her scope.

Permissions can be viewed by user, project or role.

To list the permissions of a selected user:

1. Select the option **“List / Edit User”** from the menu.
2. Navigate to a user (web-browser search functionality can be used).
3. Select the option **“Permissions”** next to the user data.



Name	Email	ID	Status	Permissions
Agustín Sánchez	agustin.sanchez@etics.es	1	ACTIVE	Permissions
Carlos Aguado Sanchez	carlos.aguado.sanchez@etics.es	2	ACTIVE	Permissions

Module	Type	Role	Action
externals	Project	ModuleAdministrator	Delete
org.etics.utilities.certificates	Component	Developer	Delete

Figure 65: How to list user permissions

List project related permissions

To list permissions of a selected user by project:

1. Select the option **“List Projects”** from the menu.
2. Navigate to a project (web-browser search functionality can be used).
3. Select the option **“List user permissions for this project”** at the bottom of the project info box.



Project Name	ID	Repository	Create date	Modify date	Vendor	Homepage	Users
org.etics	1	http://etics.es	Wed Mar 22 09:51:42 CET 2006	Mon Sep 25 00:33:09 CEST 2006	ETICS	http://www.etics.es	List user permissions for this project

Module	Type	User	Role	Action
org.etics	Project	Agustín Sánchez	ModuleAdministrator	Delete
org.etics	Project	Carlos Aguado Sanchez	Integrator	Delete

Figure 66: How to list project related permissions

List permissions for a given role

To list permissions of a selected user for a given role:

1. Select the option **“List Roles”** from the menu.
2. Navigate to a role.
3. Select the option **“List users with this role”** next to the role name.

Role List		
Name	Display Name	Description
Administrator	Administrator	This user can perform all the operations on the system including security-related operations
Integrator	Integrator	This user can submit builds and tests jobs

Users with role Integrator				
Module	Type	User	Role	Action
org.etics	Project	Gerard Boudry Gerard.Boudry@cea.fr, Gerard.Boudry@cea.fr	Integrator	Delete
org.etics	Project	Guillaume Richard Guillaume.Richard@cea.fr, Guillaume.Richard@cea.fr	Integrator	Delete

Figure 67: How to list permissions for a given role

Remove user permission

To remove user permissions, first list permissions as described in the above section, find the permission you want to remove and then select the option “Delete”.

Note that permissions are removed immediately, **without asking for a confirmation**.

Example of removing a role from user:

User List		
Name: Raymond Douglas Email: raymond.douglas@cea.fr ID: Raymond.Douglas@cea.fr	Raymond Douglas Raymond.Douglas@cea.fr	ACTIVE Permissions Delete User
Name: Carlos Aguado Sanchez Email: Carlos.Aguado.Sanchez@cea.fr ID: Carlos.Aguado.Sanchez@cea.fr	Carlos Aguado Sanchez Carlos.Aguado.Sanchez@cea.fr	ACTIVE Permissions Delete User

User permissions				
Carlos Aguado Sanchez Carlos.Aguado.Sanchez@cea.fr, Carlos.Aguado.Sanchez@cea.fr, Carlos.Aguado.Sanchez@cea.fr				
Module	Type	Role	Action	
externals	Project	ModuleAdministrator	Delete	
org.etics.utilities.certificates	Component	Developer	Delete	

User permissions				
Carlos Aguado Sanchez Carlos.Aguado.Sanchez@cea.fr, Carlos.Aguado.Sanchez@cea.fr, Carlos.Aguado.Sanchez@cea.fr				
Module	Type	Role	Action	
externals	Project	ModuleAdministrator	Delete	

Figure 68: How to remove user permissions

15.2. Administration with the Command-Line Client

This section describes in how to register users and assign roles to registered users for specific modules and configurations.

15.2.1. Manage users

How to edit a user

The ETICS command to edit a user is called `etics-user`. The syntax of the command is

```
etics-user <operation> [<options>] <dn-value>
```

The following table shows the options:

Table 28: Options for `etics-user` command

Option	Description
-h, --help	Show the usage instructions
--fname <firstname>	Specify the first name of the user. It is mandatory with the operation add.
--lname <lastname>	Specify the last name of the user. It is mandatory with the operation add.
--email <email address>	Specify the email address of the user. It is mandatory with the operation add.

`--dn <dn>`

Specify the dn information of the user certificate. It is mandatory to change the dn value during the operation modify.

`<dn-value>` represents the value of the distinguished name.

How to add a user

The command to create a user is:

```
etics-user add --fname <firstname> --lname <lastname> --email <email address> <dn-value>
```

How to modify a user

The command to modify a user is:

```
etics-user modify --fname <firstname> <dn-value>
```

If you need to change the current dn, you can run the following command:

```
etics-user modify --dn <new-dn-value> <dn-value>
```

How to remove a user

The command to remove a user is:

```
etics-user remove <dn-value>
```

15.2.2. Manage user permissions

As mentioned earlier, permissions in the ETICS system are relations between users, ETICS modules/configurations and roles.

How to assign a role

The ETICS command to edit a user is called `etics-role`. The syntax of the command is

```
etics-role <operation> [options] [<options>] <role-value>
```

The following table shows the options:

Option	Description
<code>-h, --help</code>	Show the usage instructions
<code>-m, --module <module-name></code>	Specify the module name
<code>-c, --configuration <configuration-name></code>	Specify the configuration name. In this case it is mandatory to specify also <code>--module, -m</code> option
<code>--dn <dn></code>	Specify the dn information of the user certificate

`<role>` represents the type of role handled in the ETICS infrastructure. It can be one of the following values:

- *Administrator*
- *Developer*
- *Guest*
- *Integrator*
- *ModuleAdministrator*
- *ReleaseManager*
- *Tester*

See section 1.3 (“Authentication, Authorization and Roles”) for a description of the different roles supported by ETICS.

How to add a role

The command to add a role to a user is:

```
etics-role add -m <module-name> --dn <dn-value> <role-value>
etics-role add -m <module-name> -c <configuration-name> --dn <dn-
value> <role-value>
```

How to remove a role

The command to remove a role from a user is:

```
etics-role remove -m <module-name> --dn <dn-value> <role-value>
etics-role remove -m <module-name> -c <configuration-name> --dn <dn-
value> <role-value>
```

16. CLIENT PLUGIN FRAMEWORK

This section provides an overview of the client-side plugin framework users can use to extend their build and test procedures. A recipe is also provided for users wanting to create their own plugins.

16.1. Overview

The plugin framework provides the ability for common actions to be performed during build and test execution, without putting the burden on the user for selecting which build actions should take place and when. In other words, the framework provides a clean separation between specific build and test execution, analysis and metrics collection. This section provides a high-level description of the plugin framework architecture and guiding principals.

The reason for exposing the framework to users is that the framework also provides the ability for the ETICS client to be extended dynamically, without having to modify the core client functionality. Further, ETICS users can leverage the extensibility, such that they can better reuse investments in home grown and/or commercial tools and techniques valuable to them during build and test.

In order to even better leverage the work performed by existing plugins, the plugin framework also allow plugins to *contribute* to other plugins, without having the plugin being contributed to needing intimate knowledge from the contributing plugins. For example, a plugin calculating code coverage during execution can contribute to a testing plugin, without the testing plugin requiring intimate knowledge about how code coverage is calculated.

The goal of the plugin framework can be summarised as follows:

- Provide an extendible framework for common actions to be performed during build and test execution
- Provide natural separation between specific build and test execution, analysis and metrics collection
- Provide open interface for users to register their own plugins to be executed during build and test
- Provide raw data for build and test and repository services

To guide the definition of the framework, here are examples of actions that could map to plugins, to be executed by the framework:

- **Static analysis**
 - Single line of code count (SLOC)
 - Cyclomatic complexity
 - Depth of inheritance
 - IPv6 compliance
 - Check style
- **Dynamic analysis**
 - Code coverage
 - Memory leaks
- **Test execution**
 - Java: JUnit
 - C++: CPPUNIT (CPPTEST)

- Python: PyUnit
- Script/Executable

The guiding principal in the examples above is that none of these actions related to the specifics of the modules on which these actions are performed.

Figure 69 describes the high-level framework architecture.

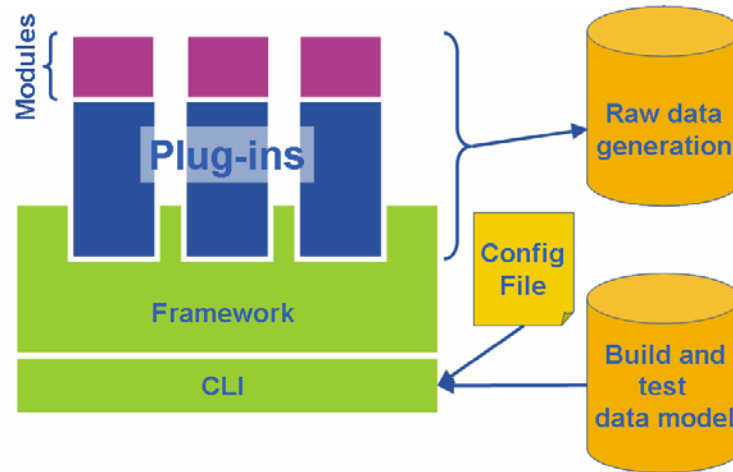


Figure 69 - Plugin Framework High-level architecture

The framework is implemented as an extension to the ETICS command-line client. In this respect the framework is totally transparent to the user. The framework also leverages the property processing already performed by the client. Raw data can be generated and should be placed in the *reports* directory in order for these generated outputs being available after the remote execution.

When a command is executed, the plugin framework first loads and registers all plugins. The registration process requires for each plugin to provide information regarding the *profile* for which they should be activated, as well as which ETICS target of which command they apply to (i.e. *VCS Command*, *Build Command* and/or *Test Command*). In other words, the following combination dictates the execution of each plugin: *profile+command+target*.

To add an extra level of flexibility, wildcards can be used by plugins during the registration. For example, the generic *SystemCallPlugin* responsible for executing as a system call the string value of each command might register for: `profiles="*", commands="*" and target="*"`.

The next section provides an overview of the specification of plugins.

16.2. Plugin specification

In order to qualify as a valid plugin, plugin implementations must comply with a few basic rules. The plugin must be provided in the form of a Python *module* or *package*. The module must be dropped in

the `plugins` directory. If a plugin is implemented as a Python module (e.g. applicable for very simple plugins), it must follow this naming convention:

- “*plugin.py”
- “plugin*.py”

On the other hand, if a plugin is implemented as Python package, the package directory must follow this naming convention:

- “*plugin”
- “plugin*”

Note: these conventions are case insensitive.

The module or the `__init__.py` module in the case of a package must implement the method `getPluginInstance`, which returns an instance of the plugin.

Python doesn't support natively the concept of *interface*, like Java or C#. Instead, each plugin must provide a class that implements the following methods:

- `register`: defines which *profile*, *command* and *target* it registers for. A helper class is available to the plugin for implementing this easily.
- `contribute`: defines which plugin it contributes to. A helper class is available to the plugin for implementing this easily.
- `init`: initialise the plugin and its environment
- `execute`: execute the plugin
- `stopExecute`: stop the execution of the plugin
- `publish`: harvest and publish to a known location the results of the execution
- `finalise`: executed once at the end of the processing, gives a chance to the plugin to clean its resources and perform any final actions, if any.

The first two methods are called as part of the registration process, in two distinct passes, which guarantees that all plugins have been registered before contributions can be made.

During each target execution (e.g. *init*, *checkstyle*, *compile*, etc in the case of the *BuildCommand*), the following methods are called for the plugins matching the current processing state: `init`, `execute`, `stopExecute` and `publish`. Each plugin is expected to call its contributors in each method. A helper class is provided by the plugin framework to easily execute the contributors.

The plugins are also provided with an instance of the `PropertyManager` class (providing full access to the property space of the current module being executed) and the `BuildStatus` class (providing access to the build and test status XML document).

Examples of plugin implementations are provided in Appendix A.

A. PLUGIN SAMPLES

The following samples show examples of plugins. The samples have been edited for clarity and are not complete. You can find a complete version in the ETICS CVS repository. Specific links are provided for each sample.

A.1. SLOCCOUNT PLUGIN

This plugin uses David Wheeler *SLOCCount* program to calculate SLOCS for a build configuration. It also generates a custom report that is included as part of the metrics overall report. An example of the output of this plugin can be seen in section 14.2. This plugin is provided as a package, which means that a `__init__.py` module is present and both files are located in a directory called *SloccountPlugin*.

The complete plugin²⁶ can be found in the ETICS CVS repository.

SloccountPlugin.py:

```
#!/usr/bin/env python
import sys
import os

import log4py

import util
import shutil
import processUtil
import Store

import PluginBase
import PluginManager
from PluginExecutionError import PluginExecutionError

logger = log4py.Logger().get_instance()
```

²⁶ <http://etics.cvs.cern.ch/cgi-bin/etics.cgi/org.etics.plugins.sloccount/>

```
class SloccountPlugin(PluginBase.PluginBase):

    def __init__(self, verbose=False):
        super(SloccountPlugin, self).__init__(verbose)
        if self.verbose: print 'Instantiating SloccountPlugin'
        self.pluginManager = None

        ...

    def register(self, pluginManager):
        # This plugin should run if no other plugins do
        profiles = ['*']
        targets = {
            'buildCommands': ['precheckstyle']
        }
        definition = PluginManager.PluginDefinition(profiles, targets)
        self.pluginManager = pluginManager

        ...

    def execute(self, profile=None, command=None, target=None, targetString=None, propertyManager=None, buildStatus=None, cmds={}, **kw):
        if propertyManager.properties.has_key('nocheckstyle'):
            if propertyManager.properties['nocheckstyle']:
                return True

            if not os.getenv('HOME'):
                os.environ['HOME'] = propertyManager['workspaceDir']

        self.outdata = None
        self.errdata = None
        if self.verbose: print 'Executing SloccountPlugin'
        process = processUtil.ProcessUtil()
        process.nooutput = not self.verbose

        cmd = "sloccount %s" % propertyManager['moduleDir']
        print '    [%s]: %s' % (target, cmd)
        try:
            self.outdata, self.errdata, err = process.systemCall(cmd)
        except RuntimeError, ex:
            if ex.args[1].find('SLOC total is zero') != -1:
                self.outdata = ex.args[1]
                return True
```

```
        else:
            logger.error(str(ex))
            raise PluginExecutionError(str(ex))
    if err != 0:
        logger.error(str(ex))
        raise PluginExecutionError((outdata, errdata, err))

    return True

def publish(self, profile=None, command=None, target=None, targetString=None, propertyManager=None, buildStatus=None, **kw):
    wasSuccessful = True
    if self.verbose: print 'Publishing'
    if propertyManager != None:
        if propertyManager.properties.has_key('nocheckstyle'):
            if propertyManager.properties['nocheckstyle']:
                return True
    else:
        return wasSuccessful

    try:
        ...

    except KeyboardInterrupt:
        raise
    except Exception, ex:
        logger.error(str(ex))
        print ex
        raise PluginExecutionError(ex)
    return wasSuccessful

def finalise(self, profile=None, command=None, target=None, targetString=None, propertyManager=None, buildStatus=None, **kw):
    wasSuccessful = True
    if self.verbose: print 'Finalising sloccount plugin'
    if propertyManager.properties.has_key('nocheckstyle'):
        if propertyManager.properties['nocheckstyle']:
            return True

    ...

    if self.verbose: print 'Generating sloccount report'
    metrics = buildStatus.CreateMetrics('SLOCCount')
```

```
metrics.detailshtmllink = 'sloccount/index.html'
for lang in self.totalLangSLOCs:
    metrics.values[lang] = self.totalLangSLOCs[lang]
metrics.value = self.totalSLOCs
buildStatus.setOverallMetrics(metrics)

return wasSuccessful
```

`__init__.py`:

```
#!/usr/bin/env python

import SloccountPlugin

def getPluginInstance():
    return SloccountPlugin.SloccountPlugin()
```

A.2. PYUNIT AND PYCOVERAGE PLUGINS

This example features a standard plugin “PyUnit” and a *contributing* plugin “PyCoverage”. The first plugin registers for a given set of commands, targets and profile, while the second plugin contributes the first one by name. In order for the plugin being contributed to accept a contribution, it must do a little more work, in order to define when in its execution it invites contributions.

The complete PyUnit²⁷ and PyCoverage²⁸ plugins and can be found in the ETICS CVS repository.

`PyUnitPlugin.py`:

²⁷ <http://etics.cvs.cern.ch/cgi-bin/etics.cgi/org.etics.build-system.plugin-framework/src/plugins/PyUnitPlugin.py>

²⁸ <http://etics.cvs.cern.ch/cgi-bin/etics.cgi/org.etics.build-system.plugin-framework/src/plugins/PyCoveragePlugin>

```
#!/usr/bin/env python

import sys
import os
import traceback

import log4py

import PluginBase
import PluginManager
from PluginManager import PluginExecutionError
import testtools.TestManager as TestManager
import unittest

logger = log4py.Logger().get_instance()

def getPluginInstance():
    return PyUnitPlugin()

class PyUnitPlugin(PluginBase.PluginBase):

    def __init__(self, verbose=True):
        super(PyUnitPlugin, self).__init__(verbose)
        if self.verbose: print 'Instantiating PyUnitPlugin'
        self.pluginManager = None
        self.pluginName = self.__module__

    def register(self, pluginManager):
        # Which profile this plugin works with
        profiles = ['pyunit']
        targets = {'buildCommands': ['test'], 'testCommands': ['test']}
        definition = PluginManager.PluginDefinition(profiles, targets)
        self.pluginManager = pluginManager
        self.pluginManager.register(self.__module__, definition)

    def init(self, profile=None, command=None, target=None, targetString='', propertyManager=None, **kw):
        if propertyManager != None:
            if propertyManager.properties.has_key('notest'):
                if propertyManager.properties['notest']:
                    return True
```

```
kw[self.__module__] = self
return self.pluginManager.initContributions(self,profile,command,target,targetString,propertyManager,**kw)

def execute(self,profile=None,command=None,target=None,targetString='',propertyManager=None,**kw):

    wasSuccessful = True
    if propertyManager.properties.has_key('notest'):
        if propertyManager.properties['notest']:
            print "    Property 'notest' set, skipping tests"
            return wasSuccessful

    ...

    # Execute any contribution made by other plugins to this one
    self.pluginManager.executeContributions(self,profile,command,target,targetString,propertyManager,**kw)

    ...

    return wasSuccessful

def stopExecute(self,profile=None,command=None,target=None,targetString=None,propertyManager=None,**kw):
    if propertyManager.properties.has_key('notest'):
        if propertyManager.properties['notest']:
            return True
    return self.pluginManager.stopExecuteContributions(self,profile,command,target,targetString,propertyManager)

def publish(self,profile=None,command=None,target=None,targetString=None,propertyManager=None,**kw):
    if self.verbose: print 'Generating test report'
    if propertyManager.properties.has_key('notest'):
        if propertyManager.properties['notest']:
            return True
    oldDir = self._changeDir(propertyManager)
    wasSuccessful = False
    try:
        generator = TestManager.ReportGenerator()
        wasSuccessful = generator.generateReport()
    finally:
        kw[self.__module__] = self
        self.pluginManager.publishContributions(self,profile,command,target, targetString,propertyManager)
        os.chdir(oldDir)
    return wasSuccessful
```

PyCoveragePlugin.py:

```
#!/usr/bin/env python
'''
Copyright (c) Members of the ETICS Collaboration. 2006.
http://www.eu-etics.org

File: PyCoveragePlugin.py

Authors: Marc-Eliaen Begin <Marc-Eliaen.Begin@cern.ch>

Version info: $Id: PyCoveragePlugin.py,v 1.13 2007/01/26 11:22:48 mbegin Exp $
'''

import sys
import os
import traceback

import log4py
import coverage

import PluginBase
import PluginManager
import testtools.TestManager as TestManager

logger = log4py.Logger().get_instance()

def getPluginInstance():
    return PyCoveragePlugin()

class PyCoveragePlugin(PluginBase.PluginBase):

    def __init__(self, verbose=False):
        super(PyCoveragePlugin, self).__init__(verbose)
        if self.verbose: print 'Instantiating PyCoveragePlugin'
        self.pluginManager = None
```



```
def register(self,pluginManager):
    self.pluginManager = pluginManager
    self.pluginManager.register(self.__module__)

def contribute(self,pluginManager):
    self.pluginManager = pluginManager
    self.pluginManager.contribute(self.__module__, 'plugins.PyUnitPlugin')

def init(self,profile=None,command=None,target=None,targetString=None,propertyManager=None,buildStatus=None,**kw):
    if self.verbose: print 'Initialising coverage for plugin: %s' % kw['plugins.PyUnitPlugin'].__module__
    return True

def execute(self,profile=None,command=None,target=None,targetString=None,propertyManager=None,buildStatus=None,**kw):
    if self.verbose: print 'Calculating coverage'
    if propertyManager != None:
        if propertyManager.properties.has_key('nocoverage'):
            if propertyManager.properties['nocoverage']:
                if self.verbose: print "Property 'nocoverage' set, skipping coverage calculation"
                return True
    coverage.start()
    return True

def stopExecution(self,profile=None,command=None,target='',targetString=None,propertyManager=None,buildStatus=None,**kw):
    if self.verbose: print 'Stopping calculating coverage'
    if propertyManager != None:
        if propertyManager.properties.has_key('nocoverage'):
            if propertyManager.properties['nocoverage']:
                return True
    coverage.stop()
    return True

def publish(self,profile=None,command=None,target='',targetString=None,propertyManager=None,buildStatus=None,**kw):
    if self.verbose: print 'Generating coverage report'
    srcDir = os.path.join('..','..','src')
    if propertyManager != None:
        if propertyManager.properties.has_key('nocoverage'):
            if propertyManager.properties['nocoverage']:
                if self.verbose: print "Property 'nocoverage' set, skipping coverage calculation"
                return True
        if propertyManager.properties.has_key('srcDir'):
            srcDir = propertyManager.properties['srcDir']
    print '    Calculating coverage for all Python modules in directory structure:'
```

```
print '    %s' % srcDir, '(this may take several minutes)'
modules = []
try:
    for n,m in sys.modules.items():
        try:
            if not 'etics' in m.__file__ or 'externals' in m.__file__:
                continue
        except AttributeError:
            continue
        if self.verbose: print 'Calculating coverage for module: %s' % n
        try:
            coverage.analysis(m.__file__)
            modules.append(m)
        except (coverage.CoverageException, SyntaxError), ex:
            if self.verbose: print 'Error analysing module %s' % m.__file__
        if self.verbose: print 'Generating coverage report for all loaded modules'
        metrics = coverage.report(modules, ignore_errors=1, show_missing=0, metrics=buildStatus.CreateMetrics('Coverage'))
        buildStatus.setOverallMetrics(metrics)
    except KeyboardInterrupt:
        raise
    except Exception, ex:
        print >> sys.stderr, 'Error occurred calculating coverage, see log for details'
        error = '\n'.join(traceback.format_exception(ex.__class__.__name__, ex, sys.exc_info()[2]))
        if self.verbose: print >> sys.stderr, error
        logger.error(error)
        return False
    return True
```

B. PROPERTIES

The following table lists all the properties that have a special semantic for the ETICS CLI. This table also specifies if the different properties are automatically created by the system, or can be defined by users, with the corresponding semantic.

Property Name	Description	Default Value
build.root	Relative to the root directory of the module, the directory from where to perform the build	<empty string>
distDir	Directory in which built artefacts are located	Dist
eticsHome	Directory of the root installation of the ETICS	The value of the ETICS_HOME environment variable if it is set, otherwise 'etics', assuming that the client was installed in the workspace
gcc.version	Version of the GCC compiler	Locally detected
libtool.build-prefix	String to be used for relocation of libtool *.la files. By default the client looks for the string '/opt/<packageName>' and replaces it with the local installation path in the ETICS workspace. If the package has a different prefix, this property must be set accordingly	/opt/<packageName>
location	Directory pointing to the location of the binaries of the configuration	The binaries path
package.forceSource	If this property is set to True, the configuration is built from source irrespective of what command-line options have been used	Undefined
package.preserve.libtool	Setting this property to True prevents the client from stripping the libtool *.la files when creating binary packages. Use with care since it may cause the packages not to be relocatable or even usable	False
package.userspec	Location (relative to the module root) of a user-defined spec file for creating RPMs	Undefined
package.tgzLocation	The default location (relative to the module root) where generated	tgz

	tarballs are stored in each module at the end of a build and where the Publisher looks for them	
package.SDEBSLocation	The default location (relative to the module root) where generated source dsc are stored in each module at the end of a build and where the Publisher looks for them	debs
package.DEBSLocation	The default location (relative to the module root) where generated binary debs are stored in each module at the end of a build and where the Publisher looks for them	debs
package.prefix	The default installation prefix for the generated RPMS	/opt/<packageName>
package.buildarch	It can be used to override the default specifier. It is necessary to set it to 'noarch' to generate 'noarch' RPMS and 'all' debs	The local platform name
package.provides	The list of Provides entries for the RPM spec file (overrides the autodetected list) or deb control file (for virtual packages)	Undefined
package.requires	The list of Requires entries for the RPM spec file (overrides the autodetected list)	Undefined
package.depends	The list of Depends entries for the deb control file (overrides the autodetected list)	Undefined
package.predepends	The list of Pre-Depends entries for the deb control file	Undefined
package.obsoletes	The list of Obsoletes entries for the RPM spec file	Undefined
package.replaces	The list of Replaces entries for the deb control file	Undefined
package.conflicts	The list of Conflicts entries for the RPM spec file and the deb control file	Undefined
package.suggests	The list of Suggests entries for the deb control file	Undefined
package.recommends	The list of Recommends entries for the deb control file	Undefined

package.enhances	The list of Enhances entries for the deb control file	Undefined
package.autoreqprov	Can be 'yes' or 'no'. If it is set to yes, rpm will try to automatically detect requires and provides from the packaged files and libs, otherwise will only use specified entries (from the ETICS packager or user defined package.provides/package.requires properties)	Yes
packageName	Name to be given to generated packages	<moduleName>
platformName	Name of the local platform	Locally detected
profile	Profile used to steer the selection of plugins	Undefined
python.version	Version of the local Python installation	Locally detected
reportsDir	Location of the reports directory	<workspaceDir>/reports
repositoryDir	Location of the repository directory	<workspaceDir>/repository
src.location	If a given module is checked-out from source, this property points to the root of the module	The location of the source files
stageDir	Directory where files are staged	<workspaceDir>/stage
test.root	Relative to the root directory of the module, the directory from where tests should be executed	<empty string>
workspaceDir	Location of the workspace	<local user-defined path>
package.createsource	Instructs the client to create source packages as well as binary packages (like the --createsource option)	False
package.nodistname	Do not insert distribution name in the package file name (e.g. slc4)	Undefined
package.usetimestamp	Append timestamp to package file name	Undefined
package.versioneddeps	Use version constraints in spec file Requires directives (>=)	False
package.strictversioning	Use strinct version constraints in spec file Requires directives (=) (must be used together with package.versioneddeps)	False
package.description	String to be used as package	<moduleDescription>

	description in the spec file	
package.summary	String to be used as package summary in the spec file	<configuration description or module name>
package.xxx.name	If the generated binary file with extension xxx has a name not following the standard conventions, setting this property will allow the client to use the actual name when creating package list reports	Undefined
package.sourcefile	Location of a source tarball to be used to create files lists for the spec file (setting this property overrides using the client generated source tarball)	Undefined
package.binfile	Location of a binary tarball to be used to create files lists for the spec file (setting this property overrides using the client generated binary tarball)	Undefined
package.customtag	Custom tag (string) to append to the generated package file name	Undefined
package.group	Value of the RPM spec file Group tag	'Unknown'
package.packager	Value of the spec file Packager tag and deb control file Maintainer tag	'ETICS'
package.define	Comma-separated list of additional %define directives for the RPM spec file. Each entry is used to add a directive like: %define <entry>	Undefined
package.priority	Value of the Priority entry in the deb control file	optional
Package.section	Value of the Section entry in the deb control file	devel
package.configFiles	Comma-separated list of files to be prefixed with the %config directive in the package spec file or added to the deb conffiles file. The entries are the full file path relative to the module root	Undefined
package.configFilesNoreplace	Comma-separated list of files to be prefixed with the	Undefined

	%config(noreplace) directive in the package spec file or added to the deb conffiles file. The entries are the full file path relative to the module root	
--	--	--

Table 29: Special properties